IBM

IBM Application System/400

SC09-1814-00

System/38-Compatible COBOL User's Guide and Reference

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page xv.

First Edition (June 1994)

This edition applies to the System/38-Compatible feature of the IBM* ILE* COBOL/400* licensed program, (Program 5763-CB1), Version 3 Release 0 Modification 5, and to all subsequent releases and modifications until otherwise indicated in new editions. Make sure you are using the proper edition for the level of the product.

Order publications through your IBM representative or the IBM branch serving your locality. Publications are not stocked at the address given below.

A form for readers’ comments is provided at the back of this publication. If the form has been removed, you may address your comments to:

IBM Canada Ltd. Laboratory
Information Development
2G/345/1150/TOR
1150 Eglinton Avenue East,
North York, Ontario, Canada M3C1H7

You can also send your comments by facsimile (attention: RCF Coordinator), or you can send your comments electronically to IBM. See "Communicating your Comments to IBM" for a description of the methods. This page immediately precedes the Readers' Comment Form at the back of this publication.

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices . . . xv
Trademarks and Service Marks . . . xv

About This Manual . . . xvii
Who Should Use this Manual . . . xvii
What You Should Know . . . xvii
Extensions . . . xviii

Chapter 1. Introduction . . . 1
General Description . . . 1
The Major Features of System/38-Compatible COBOL . . . 1
Language Level Supported by System/38-Compatible COBOL . . . 1
Federal Information Processing Standard Flagger . . . 2
How the COBOL Program is Organized . . . 4
The COBOL Divisions . . . 5
Section and Paragraphs . . . 5
Entries and Sentences . . . 5
Clauses and Statements . . . 5
Phrases . . . 6
Characters and Character Strings . . . 6
Characters . . . 6
Character Strings . . . 7
Literals . . . 7
COBOL Words . . . 8
PICTURE Character Strings . . . 12
Comments . . . 12
Separators . . . 13
Format Notation . . . 14
Methods of Referencing Data and Procedures . . . 15
Qualification . . . 15
Qualification Rules . . . 17
Subscripting and Indexing . . . 18
Identifier . . . 18
Condition-Name . . . 19
Explicit and Implicit References . . . 19
Data Attribute Specification . . . 19
Procedure Division Data References . . . 19
Transfers of Control . . . 20

Chapter 2. Coding and Entering Your Program . . . 21
System/38 Environment . . . 21
Entering the Source Program into the System . . . 21
Using SEU and the System/38-Compatible COBOL Syntax Checker to Enter Source Programs . . . 22
Standard COBOL Format Used When Entering Code . . . 24
Sequence Numbers (Columns 1-6) . . . 24
Continuation Area (Column 7) . . . 25
Area A (Columns 8-11) and Area B (Columns 12-72) . . . 25
Special Considerations . . . 27
Division Header . . . 27

Section Header . . . 27
Paragraph Header, Paragraph-Name . . . 28
Data Division Entries . . . 28
DECLARATIVES and END DECLARATIVES . . . 28
Program Spacing . . . 28
Indentation . . . 28
Continuation of Lines . . . 29
Comment Lines . . . 29
Debugging Lines . . . 29
Blank Lines . . . 30
Overall Punctuation Rules . . . 30
Identification Division . . . 30
Environment Division . . . 30
Data Division . . . 30
Procedure Division . . . 30
COPY Statement . . . 30
REPLACING Phrase . . . 32
COPY Statement Example . . . 34

Chapter 3. Compiling Your Program . . . 37
Compiler Options . . . 37
Create COBOL Program Command . . . 37
Completing the CRTCBLPGM Prompt Screens . . . 38
PROCESS Statement . . . 46
Batch Compiles . . . 47
Using COPY within the PROCESS Statement . . . 48
Using SEU to Browse through a Compiler Listing . . . 48
Compiler Output . . . 49
Command Summary . . . 49
Listing of Compiler Options . . . 49
Source Listing . . . 50
Verb Usage by Count Listing . . . 52
Data Division Map . . . 52
FIPS Messages . . . 55
Cross-Reference List . . . 56
Messages . . . 57

Chapter 4. Running and Debugging Your Program . . . 59
If the Program Ends Abnormally . . . 60
Requesting Data from Job Stream . . . 60
Debugging Your Program . . . 60
Using a Test Library . . . 61
Testing a Program . . . 62
Running a Program Normally . . . 62
Using the Same Program in Several Jobs . . . 63
Using Breakpoints . . . 63
Example of Using Breakpoints . . . 63
Considerations for Using Breakpoints . . . 69
Using a Trace . . . 71
Example of Using a Trace . . . 72
Considerations for Using a Trace . . . 73
Using a Debug Run-Time Switch . . . 74
File Status . . . 74
Using a COBOL Formatted Dump . . . 74

Reply Modes and System Reply List . . . 75
Example of Using a Dump . . . 75
Identifying COBOL Problems . . . 86
Calling for Help . . . 88

Chapter 5. Interactive Processing Considerations and Example Programs 89
Externally Described Transaction File . . . 90
Processing an Externally Described TRANSACTION File . . . 92
Indicators . . . 92
Indicators in the Record Area . . . 93
Indicators in a Separate Indicator Area . . . 93
ASSIGN Clause with Separate Indicator Area Attribute . . . 93
Data Description Entry–Boolean Data . . . 94
Special Considerations . . . 94
INDICATORS Phrase . . . 95
Indicators in the Record Area . . . 95
Indicators in a Separate Indicator Area . . . 96
Indicators Example Programs . . . 96
Subfiles . . . 106
Use of Subfiles . . . 108
Multiple Device Files and Single Device Files . . . 112
Program Described Transaction Files . . . 122
Environment Division . . . 122
File-Control Entry . . . 122
ASSIGN Clause . . . 122
ORGANIZATION Clause . . . 123
ACCESS MODE Clause . . . 123
RELATIVE KEY Clause . . . 124
FILE STATUS Clause . . . 124
CONTROL-AREA Clause . . . 124
Data Division . . . 125
File Description Entry . . . 125
Boolean Data Facilities . . . 126
Procedure Division . . . 126
ACCEPT Statement . . . 126
Attribute Data Formats . . . 127
ACQUIRE Statement . . . 127
CLOSE Statement . . . 128
DROP Statement . . . 129
OPEN Statement . . . 129
Common Processing Facilities . . . 130
FORMAT Phrase . . . 130
DB-FORMAT-NAME Special Register . . . 130
INDICATORS Phrase . . . 130
SUBFILE Phrase . . . 131
TERMINAL Phrase . . . 131
READ Statement . . . 131
REWRITE Statement . . . 138
WRITE Statement . . . 140
USE Statement . . . 145
Work Station Example Programs . . . 145

Chapter 6. Example Programs . . . 185
Sequential File Creation . . . 185

Sequential File Updating and Extension . . . 188
Indexed File Creation . . . 190
Indexed File Updating . . . 192
Relative File Creation . . . 197
Relative File Updating . . . 199
Relative File Retrieval . . . 201

Chapter 7. System/38-Compatible COBOL Programming Considerations 205
Device Independence/Device Dependence . . . 206
Spooling . . . 207
Output Spool . . . 208
Input Spool . . . 208
System Override Considerations . . . 209
File and Record Locking by COBOL . . . 209
Releasing a Record Read for Update . . . 210
Unblocking Input Records and Blocking Output Records . . . 210
Externally Described/Program Described Files . . . 211
COBOL Specifications for Externally Described Files . . . 216
Format 2 COPY Statement, DDS or DD Formats . . . 219
Data Structures Generated . . . 221
Format (Record) Level Structures . . . 221
Data Field Structures . . . 222
Indicator Structures . . . 223
INDICATOR Attribute of the Format 2 COPY Statement . . . 223
Generation of I-O Formats . . . 224
Redefinition of Formats . . . 225
Additional Notes on Field and Format Names . . . 225
Floating Point Fields . . . 226
Considerations when Using REPLACING with Format 2 COPY Statement 226
Access Path . . . 227
Record Keys and Common Keys . . . 228
Overriding or Adding COBOL Functions to the External Description . . 228
Level Checking . . . 229
Program Described Files . . . 229
Specific COBOL File Processing . . . 229
Printer File Considerations . . . 229
SPECIAL-NAMES Paragraph and the ADVANCING Phrase . . . 230
LINAGE Clause . . . 230
FORMATFILE Files . . . 231
DISK and DATABASE File Considerations . . . 236
DATABASE versus DISK Files . . . 236
Processing Methods for DISK and DATABASE Files . . . 236
COBOL Indexed Files . . . 236
Referring to a Partial Key . . . 237
Logical File Considerations . . . 240
COBOL Relative File . . . 242
COBOL Sequential File . . . 243
COBOL File Organization and AS/400 File Access Path Considerations 244
File Processing Methods . . . 244
Descending File Considerations . . . 246
Commitment Control Considerations . . . 247
Performance Considerations . . . 255
Segmentation . . . 255
Debugging . . . 255

*NORANGE Option . . . 255
Indicators . . . 255
Commitment Control . . . 255
Program Loops . . . 255
Tracing a Loop in a Program . . . 255
Errors That Can Cause a Loop . . . 256
Exceptions and Some of Their Causes . . . 256
Recovery after a Failure . . . 257
Recovery with Commitment Control . . . 257
Transaction File Recovery . . . 257
Inter-Program Communication Considerations . . . 262
Return of Control From a Called Program . . . 262
Initialization of Storage . . . 262
Local Data Area . . . 266
File Considerations . . . 266

Chapter 8. Identification and Environment Divisions . . . 267
IDENTIFICATION DIVISION . . . 267
Coding Example . . . 268
PROGRAM-ID Paragraph . . . 268
Other Optional Paragraphs . . . 268
ENVIRONMENT DIVISION . . . 269
Coding Example . . . 270
Configuration Section . . . 270
SOURCE-COMPUTER Paragraph . . . 271
OBJECT-COMPUTER Paragraph . . . 271
MEMORY SIZE Clause . . . 272
PROGRAM COLLATING SEQUENCE Clause . . . 272
SEGMENT-LIMIT Clause . . . 272
SPECIAL-NAMES Paragraph . . . 272
Function-Name-1 Clause . . . 272
Function-Name-2 Clause . . . 273
Coding Example . . . 274
Alphabet-Name Clause . . . 275
CURRENCY SIGN Clause . . . 277
DECIMAL-POINT IS COMMA Clause . . . 277
Input-Output Section . . . 277
Files . . . 278
Data Base Files . . . 278
Device Files . . . 278
DDM Files . . . 278
Paragraphs . . . 278
File Processing Summary . . . 279
Data Organization . . . 279
Access Modes . . . 280
Access Mode Allowed for Each File Type . . . 281
FILE-CONTROL Paragraph . . . 281
FILE-CONTROL Paragraph–General Considerations . . . 284
SELECT Clause . . . 284
ASSIGN Clause . . . 284
RESERVE Clause . . . 286
ORGANIZATION Clause . . . 286
ACCESS MODE Clause . . . 288
RECORD KEY Clause (Indexed File) . . . 289

FILE STATUS Clause . . . 291
I-O-CONTROL Paragraph . . . 292
RERUN Clause . . . 292
SAME Clause . . . 292
MULTIPLE FILE TAPE Clause . . . 293
COMMITMENT CONTROL Clause . . . 293

Chapter 9. Data Division . . . 295
Data Division Concepts . . . 295
External Data . . . 295
Internal Data . . . 295
Data Relationships . . . 296
Data Division Organization . . . 296
Coding Example . . . 297
Example Data Division Entries . . . 298
File Section . . . 298
Working-Storage Section . . . 299
Linkage Section . . . 299
File Description Entry . . . 300
File-Name . . . 303
BLOCK CONTAINS Clause . . . 303
RECORD CONTAINS Clause . . . 304
LABEL RECORDS Clause . . . 305
VALUE OF Clause . . . 306
DATA RECORDS Clause . . . 306
LINAGE Clause . . . 306
CODE-SET Clause . . . 309
Data Description . . . 309
Data Description Concepts . . . 309
Level Concepts . . . 310
Level-Numbers . . . 310
Special Level-Numbers . . . 312
Indentation . . . 312
Classes of Data . . . 312
Standard Alignment Rules . . . 313
Standard Data Format . . . 314
Character-String and Item Size . . . 314
Signed Data . . . 314
Operational Signs . . . 314
Editing Signs . . . 314
Record Description Entry . . . 314
Data Description Entry . . . 314
Format 1 . . . 316
Format 2 . . . 317
Format 3 . . . 317
Format 4–Boolean Data . . . 318
Level-Numbers . . . 318
Data-Name or FILLER Clause . . . 318
REDEFINES Clause . . . 319
USAGE Clause . . . 322
DISPLAY Phrase . . . 322
Computational Phrases . . . 323
INDEX Phrase . . . 324
SIGN Clause . . . 326

OCCURS Clause . . . 327
INDICATOR Clause . . . 327
SYNCHRONIZED Clause . . . 327
JUSTIFIED Clause . . . 327
BLANK WHEN ZERO Clause . . . 328
VALUE Clause . . . 328
General Considerations . . . 329
Format 1 Considerations . . . 330
Format 2 Considerations . . . 330
PICTURE Clause . . . 332
Symbols Used in the PICTURE Clause . . . 332
PICTURE Clause Editing . . . 338
RENAMES Clause . . . 343

Chapter 10. Procedure Division . . . 347
Procedure Division Concepts . . . 347
Declaratives . . . 347
Procedures . . . 347
Procedure Division Organization . . . 348
Categories of Sentences . . . 349
Categories of Statements . . . 349
Categories of Expressions . . . 351
Example Procedure Division Statements . . . 352
Arithmetic Expressions . . . 352
Arithmetic Operators . . . 353
Conditional Expressions . . . 354
Simple Conditions . . . 354
Class Condition . . . 354
Condition-Name Condition . . . 355
Relation Condition . . . 356
Sign Condition . . . 359
Switch-Status Condition . . . 359
Complex Conditions . . . 359
Negated Simple Conditions . . . 360
Combined Conditions . . . 360
Abbreviated Combined Relation Conditions . . . 363
Declaratives . . . 364
EXCEPTION/ERROR Declarative . . . 365
File-Name Phrase . . . 365
INPUT Phrase . . . 365
OUTPUT Phrase . . . 365
I-O Phrase . . . 365
EXTEND Phrase . . . 365
General Considerations . . . 366
Programming Notes . . . 366
Conditional Statements . . . 367
IF Statement . . . 367
Nested IF Statements . . . 368
Input/Output Statements . . . 370
Common Input/Output Phrases . . . 370
Status Key . . . 370
INVALID KEY Condition . . . 371
INTO/FROM Identifier Phrase . . . 371
Current Record Pointer . . . 371

DB-FORMAT-NAME Special Register . . . 373
ACCEPT Statement . . . 373
Format 1 Considerations . . . 374
Format 2 Considerations . . . 375
Format 3 Considerations . . . 376
Format 4 Considerations . . . 376
Format 5 Considerations . . . 376
Programming Notes . . . 376
ACQUIRE Statement . . . 377
CLOSE Statement . . . 377
COMMIT Statement . . . 381
DELETE Statement . . . 382
DISPLAY Statement . . . 385
Format 1 Considerations . . . 385
Format 2 Considerations . . . 388
DROP Statement . . . 389
OPEN Statement . . . 389
READ Statement . . . 393
REWRITE Statement . . . 402
ROLLBACK Statement . . . 406
START Statement . . . 407
WRITE Statement . . . 412
Arithmetic Statements . . . 420
Arithmetic Statement Operands . . . 420
Size of Operands . . . 420
Overlapping Operands . . . 421
Multiple Results . . . 421
Common Phrases . . . 422
CORRESPONDING Phrase . . . 422
GIVING Phrase . . . 423
ROUNDED Phrase . . . 423
SIZE ERROR Phrase . . . 423
ADD Statement . . . 424
COMPUTE Statement . . . 425
DIVIDE Statement . . . 425
MULTIPLY Statement . . . 427
SUBTRACT Statement . . . 428
Data Manipulation Statements . . . 429
INSPECT Statement . . . 429
INSPECT . . . 433
TALLYING Phrase . . . 433
REPLACING Phrase . . . 433
BEFORE/AFTER Phrases . . . 434
INSPECT Statement Examples . . . 435
MOVE Statement . . . 436
General Considerations . . . 436
Elementary Moves . . . 437
Group Moves . . . 439
Format 1 Considerations . . . 440
Format 2 Considerations . . . 440
SET Statement . . . 440
STRING Statement . . . 441
STRING Statement Processing . . . 442
STRING Statement Example . . . 443

UNSTRING Statement . . . 445
Sending Field . . . 445
Data Receiving Fields . . . 446
UNSTRING Statement Processing . . . 447
UNSTRING Statement Example . . . 450
Procedure Branching Statements . . . 452
ALTER Statement . . . 452
Segmentation Information . . . 453
EXIT Statement . . . 453
GO TO Statement . . . 454
Format 1–Unconditional GO TO . . . 454
Format 2–Conditional GO TO . . . 455
PERFORM Statement . . . 455
Format 1 . . . 458
Format 2 . . . 458
Format 3 . . . 458
Format 4 . . . 459
Varying One Identifier . . . 459
Varying Two Identifiers . . . 461
Varying Three Identifiers . . . 464
Segmentation Information . . . 466
STOP Statement . . . 467
Compiler-Directing Statements . . . 468
ENTER Statement . . . 468

Chapter 11. Using the Additional COBOL Functions . . . 469
TABLE HANDLING . . . 469
Table Handling Concepts . . . 469
Table Definition . . . 469
Table References . . . 471
Subscripting . . . 472
Indexing . . . 473
Restrictions on Subscripting and Indexing . . . 474
Table Initialization . . . 475
Data Division–Table Handling . . . 476
OCCURS Clause . . . 476
Fixed Length Tables . . . 477
Variable Length Tables . . . 477
ASCENDING/DESCENDING KEY Phrase . . . 478
INDEXED BY Phrase . . . 479
USAGE IS INDEX Clause . . . 480
Procedure Division–Table Handling . . . 480
Relation Conditions . . . 481
SEARCH Statement . . . 481
Format 1 . . . 482
Format 2 . . . 483
Programming Notes . . . 486
SEARCH Example . . . 486
SET Statement . . . 488
Format 3 . . . 489
Format 4 . . . 490
SORT/MERGE . . . 491
Sort/Merge Concepts . . . 491
Sort Concepts . . . 492

Merge Concepts . . . 492
Environment Division–SORT/MERGE . . . 492
File-Control Paragraph . . . 493
I-O-Control Paragraph . . . 493
Data Division–SORT/MERGE . . . 493
Procedure Division–SORT/MERGE . . . 494
MERGE Statement . . . 494
SORT Statement . . . 495
MERGE Statement and SORT Statement Phrases . . . 495
ASCENDING/DESCENDING KEY Phrase . . . 496
COLLATING SEQUENCE Phrase . . . 496
USING Phrase . . . 497
GIVING Phrase . . . 497
SORT INPUT PROCEDURE Phrase . . . 498
SORT/MERGE OUTPUT PROCEDURE Phrase . . . 499
SORT or MERGE INPUT/OUTPUT PROCEDURE Control . . . 500
RELEASE Statement (Sort Function Only) . . . 500
RETURN Statement . . . 501
SORT/MERGE Programming Notes . . . 501
SEGMENTATION FEATURE . . . 503
Segmentation Concepts . . . 503
Program Segments . . . 503
Fixed Segments . . . 503
Independent Segments . . . 503
Segmentation Logic . . . 504
Segmentation Control . . . 504
COBOL Source Program Considerations . . . 505
Segmentation–Environment Division . . . 505
Segmentation–Procedure Division . . . 505
Segmentation–Special Considerations . . . 506
ALTER Statement . . . 506
PERFORM Statement . . . 506
SORT and MERGE Statements . . . 506
Calling and Called Programs . . . 507
Inter-Program Communication Function . . . 507
Inter-Program Communication Concepts . . . 507
Transfers of Control . . . 507
Common Data . . . 507
COBOL Language Considerations . . . 508
Data Division–Inter-Program Communication . . . 508
Record Description Entries . . . 509
Data Item Description Entries . . . 509
Procedure Division–Inter-Program Communication . . . 509
CALL Statement . . . 510
USING Phrase . . . 511
CANCEL Statement . . . 512
EXIT PROGRAM Statement . . . 513
STOP RUN Statement . . . 514
Inter-Program Communication Function Examples . . . 514
OS/400 Graphics Support . . . 517
DEBUGGING FEATURES . . . 517
COBOL Source Language Debugging . . . 517
Compile-Time Switch . . . 518
Run-Time Switch . . . 518

USE FOR DEBUGGING Declarative . . . 521
DEBUG-ITEM Special Register . . . 523
Debugging Lines . . . 524
FIPS FLAGGER . . . 525

Appendix A. Summary of IBM Extensions . . . 527
Character-String Considerations . . . 527
Identification Division . . . 527
Environment Division . . . 527
Data Division . . . 528
Procedure Division . . . 528
COPY Statement–All Divisions . . . 530
TRANSACTION Files . . . 530
Compiler Options . . . 531

Appendix B. Associated Card File Processing . . . 533
Environment Division . . . 533
SELECT Clause . . . 533
ASSIGN Clause . . . 533
Data Division . . . 534
Procedure Division . . . 534

Appendix C. Intermediate Result Fields . . . 537
Compiler Calculation of Intermediate Results . . . 538

Appendix D. COBOL Message Descriptions . . . 541
Interactive Messages . . . 541
Compilation Messages . . . 541
Severity Levels . . . 541

Appendix E. File Structure Support Summary and Status Key Values . 543
Status Key Values and Meanings . . . 548
Attribute Data Formats . . . 551
Display Device Attribute Data . . . 551
Communications Device Attribute Data . . . 551
OPEN-FEEDBACK and I-O-FEEDBACK Data Areas . . . 552
OPEN-FEEDBACK . . . 552
I-O-FEEDBACK . . . 552

Appendix F. EBCDIC and ASCII Collating Sequences . . . 555
EBCDIC Collating Sequence . . . 555
ASCII Collating Sequence . . . 559

Appendix G. COBOL Reserved Words . . . 563

Appendix H. Summary of Clauses and Statements . . . 567

Appendix I. COBOL Differences . . . 571
Features of System/38 COBOL Not Supported by System/38-Compatible COBOL . . . 571
Migrating System/38 ANSI 74 COBOL Programs to AS/400 ANSI 85 COBOL/400 . . . 571

Appendix J. Glossary of Abbreviations . . . 575

Bibliography 577

Glossary of Terms 579

Index 589

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent product, program, or service that does not infringe on any of IBM's intellectual property rights may be used instead of the IBM product, program, or service. Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 208 Harbor Drive, Stamford, Connecticut, USA 06904-2501.

Changes or additions to the text are indicated by a vertical line (|) to the left of the addition or change.

This publication contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks and Service Marks

The following terms, denoted by an asterisk (*), used in this publication, are trademarks of the IBM Corporation in the United States or other countries:

Application System/400
AS/400
GDDM
ILE
OS/400
400
APPN
COBOL/400
IBM
Operating System/400
RPG/400

About This Manual

This manual is a guide and reference for creating or working with programs intended for use on System/38 or on the AS/400 system using System/38-Compatible COBOL.

Using this manual, you will be able to:

- Design COBOL programs
- Code COBOL programs
- Enter and compile COBOL programs
- Test and debug COBOL programs
- Study coded COBOL examples and sample programs

Who Should Use this Manual

This manual is meant for programmers having some experience in writing COBOL programs, and for operators who run these programs.

When users work with System/38-Compatible COBOL programs on the AS/400 system, they can do so only by accessing an AS/400 system feature called the System/38 environment, which emulates System/38. This manual is a combination user's guide and reference document for System/38-Compatible COBOL in the System/38 environment.

What You Should Know

Before you use this manual, you should be familiar with certain information:

- Card files (such as `READER, PUNCH, and PUNCHPRINT`), card devices, and related language elements are not supported by System/38-Compatible COBOL in the System/38 environment. Programs written with references to card devices and related language elements will be syntax checked but compilation in the System/38 environment will fail. These programs are compatible with the System/38 and compilation of such programs can be performed on a System/38 on which the ILE* COBOL/400 Licensed Program (Program 5763-CB1) is installed. A note indicating that card files, card devices and related language elements are not supported in System/38-Compatible COBOL will appear where these references occur in the manual.
- The use of the term Format 2 `COPY` statement throughout this manual is intended as a reference to the `COPY` statement, DDS or DD format.
- You should be familiar with your display station (also known as a work station), and its controls. There are also some elements of its display and certain keys on the keyboard that are standard regardless of which software system is currently running at the display station, or which hardware system the display station is hooked up to. Some of these keys are:
  - Cursor movement keys
  - Command keys
  - Field exit keys
  - Insert and delete keys
  - The Error Reset key.

This information is contained in *Guide to Programming Application and Help Displays, SC41-0011*.

- You should know how to operate your display station when it is hooked up to the IBM AS/400 system and running AS/400 software. This means knowing about OS/400 and the Control Language (CL) to do such things as:
  - Sign on and sign off the display station
  - Interact with displays
  - Use Help
  - Enter control language commands
  - Call utilities
  - Respond to messages.

  To find out more about this operating system and its control language, refer to:
  - *Programming: Control Language Reference, SC41-0030*
  - *Programming: Control Language Programmer's Guide*
  - *Programming: Reference Summary, SX41-0028*
  - *System/38-Compatible COBOL Reference Summary*
- You should know how to call and use certain utilities available on the AS/400 system:
  - The Screen Design Aid (SDA) utility used to design and code displays. This information is contained in *Application Development Tools: Screen Design Aid User's Guide and Reference, SC09-1340*.
  - The Source Entry Utility (SEU), which is a full-screen editor you can use to enter and update your source members. This information is contained in *Application Development Tools: Source Entry Utility User's Guide and Reference, SC09-1338*.

Extensions

To help you, IBM provides several extensions to American National Standard (ANS) COBOL, X3.23-1974. The more significant extensions include:

- `TRANSACTION I-O`: Used to send or receive records from a work station.
- Data Base I-O: The System/38-Compatible COBOL programmer can define data as he does presently. Thus, the System/38-Compatible COBOL programmer can use standard COBOL Environment and Data Division entries to specify file identification, field definitions, and data structures. Clauses have been added to the `READ`, `WRITE`, `REWRITE`, `DELETE`, and `START` verbs to support the AS/400 data base.
- `COPY`: Support for externally described files.
- Extended data types: Computational-3 (packed decimal), and computational-4 (binary).
- Use of apostrophe instead of quotes.

Chapter 1. Introduction

General Description

COBOL (Common Business Oriented Language) is a programming language that resembles English. As its name implies, COBOL is especially suited to the processing of business problems. COBOL can be used to manipulate large files of data in a relatively simple way. That is, COBOL emphasizes the description and handling of data items and of input/output records.

The System/38-Compatible COBOL Compiler and Library is an IBM licensed program that accepts and compiles COBOL programs written in accordance with the ANS COBOL X3.23-1974 standard. This licensed program also includes a number of IBM extensions. The following sections describe the language level implemented and the language-independent compiler features.

The Major Features of System/38-Compatible COBOL

IBM System/38-Compatible COBOL provides the following features:

- Syntax-checking during compilation saves machine time while debugging source syntax errors. The source program is scanned for syntax errors and, if any serious errors are found, the associated error messages are generated, but no object program is produced.
- The sorted cross-reference option of the compiler provides a listing of each Data Division name and Procedure Division procedure-name, and indicates the statement numbers of each reference or change to the item.
- Inter-program calls allow programs written in System/38-Compatible COBOL to call or be called by other programs written in System/38-Compatible COBOL, COBOL/400*, AS/400 PL/I, RPG/400*, or in the AS/400 control language.
- Diagnostic messages below a user-specified level can be suppressed.

Language Level Supported by System/38-Compatible COBOL

When you code COBOL statements into a program, these statements can access a variety of functions that are available. Each function is associated with a program module. The table in Table 1 on page 2 identifies these modules and the version (language level) that is supported.

The notation used within Table 1 on page 2 has the following interpretation:

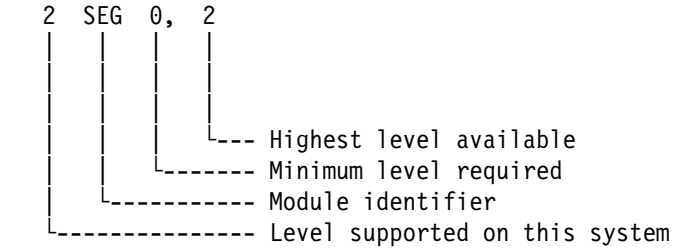


Table 1. IBM System/38-Compatible COBOL Support	
System/38-Compatible COBOL Processing Modules	**Description of the Module**
Nucleus `2 NUC 1, 2`	Contains the language elements necessary for internal processing.
Table Handling `2 TBL 1, 2`	Contains the language elements necessary for: (1) definition of tables; (2) identification, manipulation, and use of indexes; (3) reference to the items within tables. Provides the ability to define fixed-length or variable-length tables of up to three dimensions. Items in the tables can be referred to by using a subscript or an index.
Sequential I-O `2 SEQ 1, 2`	Allows definition and access of sequentially organized external files. IBM System/38-Compatible COBOL Sequential I-O provides all level 2 functions except for support of the `RERUN` clause.
Relative I-O `2 REL 0, 2`	Provides the capability of defining and accessing disk files in which records are identified by relative record numbers. A file can be accessed randomly and sequentially in the same COBOL program. IBM System/38-Compatible COBOL Relative I-O provides all level 2 functions except for support of the `RERUN` clause.
Indexed I-O `2 INX 0, 2` **Note:** Alternate key omitted.	Provides the capability of defining disk files in which records are identified by the value of a key and accessed through an access path. IBM System/38-Compatible COBOL Indexed I-O provides all level 2 functions except for support of the `ALTERNATE RECORD KEY` clause, the `RERUN` clause, and the `KEY IS` phrase of the `READ` statement.
Sort-Merge `2 SRT 0, 2`	Allows the inclusion of one or more sorts in a COBOL program and use of the merge facility.
Report Writer `0 RPW 0, 1`	Provides semiautomatic production of printed reports.
Segmentation `2 SEG 0, 2`	Provides overlaying at object time of Procedure Division sections.
Library `2 LIB 0, 2`	Allows inclusion of predefined COBOL text in a program.
Debug `2 DEB 0, 2`	Provides for user-specification of statements and procedures for debugging.
Inter-Program Communication `2 IPC 0, 2`	Provides facilities for a program to communicate with one or more other programs. Also provides capability to transfer control to another program known at compile time, and the ability for both programs to have access to certain data items.
Communication `0 COM 0, 2`	Provides the ability to access, process, and create messages or portions of messages; also provides the ability to communicate through a Message Control System with local and remote communication devices.

Federal Information Processing Standard Flagger

Depending on the compiler option chosen (NO, H, HI, LI, or L), the Federal Information Processing Standard (FIPS) Flagger identifies COBOL source statements and clauses that do not conform to the federal standard, FIPS PUB 21-1, at the specified level. 1975 FIPS COBOL, December 1975, is a compatible subset of ANS (American National Standard) COBOL, X3.23-1974. 1975 FIPS COBOL is subdivided into four levels: full, high-intermediate, low-intermediate, and low. Any program written to conform to one of these levels of 1975 FIPS COBOL must conform to one of these levels of 1975 FIPS COBOL processing.

Elements that are specified in the COBOL source program and that are not included in 1975 FIPS COBOL are flagged as described in the following sections.

Full FIPS Flagging: When flagging for the full FIPS level, messages are issued for all elements specified in the source program that are described in messages CBL85xx.

High-Intermediate FIPS Flagging: When flagging for the high-intermediate FIPS level, messages are issued for all elements specified in the source program that are described in messages CBL85xx and CBL84xx.

Low-Intermediate FIPS Flagging: When flagging for the low-intermediate FIPS level, messages are issued for all elements specified in the source program that are described in messages CBL85xx, CBL84xx, and CBL83xx.

Low FIPS Flagging: When flagging for the low FIPS level, messages are issued for all elements specified in the source program that are described in messages CBL85xx, CBL84xx, CBL83xx, and CBL82xx.

Figure 1 shows the 1974 Standard COBOL processing modules included in each of the levels of 1975 FIPS COBOL.

1974 ANS Module Name	Full FIPS	High-Intermediate FIPS	Low-Intermediate FIPS	Low FIPS
Nucleus	2 NUC 1,2	2 NUC 1,2	1 NUC 1,2	1 NUC 1,2
Table Handling	2 TBL 1,2	2 TBL 1,2	1 TBL 1,2	1 TBL 1,2
Sequential I-O	2 SEQ 1,2	2 SEQ 1,2	1 SEQ 1,2	1 SEQ 1,2
Relative I-O	2 REL 0,2	2 REL 0,2	1 REL 0,2	0 REL 0,2
Indexed I-O	2 INX 0,2	0 INX 0,2	0 INX 0,2	0 INX 0,2
Sort-Merge	2 SRT 0,2	1 SRT 0,2	0 SRT 0,2	0 SRT 0,2
Report Writer	0 RPW 0,1	0 RPW 0,1	0 RPW 0,1	0 RPW 0,1
Segmentation	2 SEG 0,2	1 SEG 0,2	1 SEG 0,2	0 SEG 0,2
Library	2 LIB 0,2	1 LIB 0,2	1 LIB 0,2	0 LIB 0,2
Debug	2 DEB 0,2	2 DEB 0,2	1 DEB 0,2	0 DEB 0,2
Inter-Program Communication	2 IPC 0,2	2 IPC 0,2	1 IPC 0,2	0 IPC 0,2
Communications	2 COM 0,2	2 COM 0,2	0 COM 0,2	0 COM 0,2

Key:

```
2 SEG 0, 2
|  |  |  |
|  |  |  └---Highest level available in 1974 ANS COBOL
|  |  └------Minimum level required by 1974 ANS COBOL
|  └---------Module identifier
└------------Level supported
```

Figure 1. 1974 American National Standard and FIPS Levels

How the COBOL Program is Organized

Every COBOL source program is organized into four divisions. These divisions are divided into a hierarchy of smaller parts. These parts parallel the structure of the English language.

Each division is composed of sections, which in turn are composed of paragraphs, as required. Paragraphs are composed of entries or sentences, depending on the division. Entries are formed from clauses; clauses themselves are formed from phrases. Sentences are formed from statements; statements also being a group of phrases. The four divisions are organized in the following way:

- The Identification Division:

  (Section)[1]
  Paragraphs
  Entries

- The Environment Division:

  Sections
  Paragraphs
  Entries
  Clauses
  Phrases

- The Data Division:

  Sections
  (Paragraphs)[2]
  Entries
  Clauses
  Phrases

- The Procedure Division:

  Sections
  Paragraphs
  Statements
  Phrases.

The elements of the COBOL program are composed of collections of character strings and the separators between the character strings.

[1] The Identification Division has only one section. It is therefore unnamed, and the division is composed only of a series of paragraphs.

[2] In the Data Division each section has only one paragraph. The paragraph is therefore unnamed, and each section is composed of a series of entries.

The COBOL Divisions

Each COBOL program division must begin with a division header and the divisions must be placed in proper sequence.

The four divisions and their functions in solving data processing problems are:

- **Identification Division**: The Identification Division names the program and allows you to optionally provide documentary information such as the date the program was written and the date it was compiled, as well as to provide other information such as authorship and security level.
- **Environment Division**: The Environment Division describes the computer(s) to be used and specifies the machine(s) and equipment features used by the program. This description defines the relationship of files of data with input/output devices.
- **Data Division**: The Data Division defines the nature and characteristics of all data the program is to process; that is, the data used in input/output operations and any data developed by the program for internal processing.
- **Procedure Division**: The Procedure Division consists of statements that are able to process the data in the manner you define. The statements are processed in the order they are placed in the program unless you define another order.

  Specific rules for the elements in these divisions are given in the reference part of this manual.

Section and Paragraphs

Each section and paragraph solves a major part of the problem for which the program was developed.

Entries and Sentences

Each paragraph is composed of:

- Entries (Identification, Environment, and Data Divisions)
- Sentences (Procedure Division).

A sentence is a series of statements ending with a period, while an entry is a series of clauses ending with a period.

Clauses and Statements

Clauses and statements have specific functions in a COBOL source program:

- A clause (Environment and Data Divisions) modifies the basic entry by adding attributes (options) to that entry
- A statement (Procedure Division) specifies an action to be taken within the larger scope of the sentence.

Phrases

A phrase is an ordered set of one or more consecutive COBOL character strings that form a portion of a COBOL clause or statement. A phrase can be used to modify the basic meaning of the clause or statement.

Characters and Character Strings

The most basic components of System/38-Compatible COBOL are the characters and character strings that form each clause and statement. Character strings have various names depending on which characters form the string or on the purpose of the string.

Characters

In System/38-Compatible COBOL, the smallest unit of data is the character. Fifty-one Extended Binary Coded Decimal Interchange Code (EBCDIC) characters form the COBOL character set:

- The 26 letters of the alphabet
- The 10 Arabic numerals
- Fifteen special characters.

Except in comments and nonnumeric literals (which may use any character within the EBCDIC set), the 51 characters are the only characters valid in a COBOL program.

IBM Extension

The apostrophe can be used in place of the quotation mark. See “Compiler Options” on page 37.

End of IBM Extension

Note: Throughout this manual, the quotation mark is used because it is the default option. In all cases, the apostrophe can be used only if the default option is overridden.

Table 2 shows these COBOL characters in ascending EBCDIC sequence and their usage in a COBOL program. These single COBOL characters are put together to form character strings and separators described in the following sections.

Table 2 (Page 1 of 2). COBOL Characters and Their Meanings

COBOL Character	Meaning	Use
	space	punctuation character
.	decimal point; period	editing character; punctuation character
<	less than sign	relation character
(	left parenthesis	punctuation character
+	plus sign	arithmetic operator; sign; editing character
$	dollar sign	editing character

Table 2 (Page 2 of 2). COBOL Characters and Their Meanings

COBOL Character	Meaning	Use
*	asterisk	arithmetic operator; editing character
)	right parenthesis	punctuation character
;	semicolon	punctuation character
–	minus sign; hyphen	arithmetic operator; sign; editing character
/	stroke or slash	arithmetic operator; editing character
,	comma	punctuation character; editing character
>	greater than sign	relation character
=	equal sign	relation character; punctuation character
" or '	quotation mark or apostrophe	punctuation character
A through Z	alphabet	alphabetic character
0 through 9	Arabic numerals	numeric character

Note: All COBOL characters are considered to be alphanumeric.

Character Strings

A character string is a character or a sequence of consecutive characters. COBOL character strings form:

- Literals
- COBOL words
- PICTURE character strings
- Comments.

Literals

A literal is a character string that has a value exactly the same as the ordered set of characters of which it is composed or by the specification of a figurative constant. The three types of literals are:

- Nonnumeric
- Numeric
- Boolean.

Nonnumeric Literals: A nonnumeric literal is a character string that can contain any character from the EBCDIC set.

IBM Extension

A nonnumeric literal can contain a maximum of 160 characters.

End of IBM Extension

You must enclose a nonnumeric literal with quotation marks (or apostrophes, if the APOST option is in effect). The enclosing apostrophes are not part of the literal.

Any punctuation characters included within a nonnumeric literal are part of the value of the literal. An embedded quotation mark must be represented by two adjacent quotation marks (" "); one quotation mark (") is then part of the value of the literal.

Every nonnumeric literal is in the alphanumeric data category. Data categories are defined under "PICTURE Clause" on page 332.

Numeric Literals: A numeric literal is a character string with characters selected from the digits 0 through 9, and, optionally, a + or - sign character and the decimal point. The following rules apply:

- One through 18 digits are allowed.
- Only one sign character is allowed. If you include a sign character, it must be the leftmost character of the literal. If the literal is unsigned, it is considered to have a positive value.
- Only one decimal point is allowed. If you include a decimal point, it is treated as an assumed decimal point (not counted as a character position in the literal). The decimal point can appear anywhere within the literal except as the rightmost character. If the literal contains no decimal point, it is considered an integer. The word *integer* appearing in a format represents a numeric literal of nonzero value that contains no sign and no decimal point; any other restrictions are included with the description of the format.

The value of a numeric literal is the number expressed by the characters in the literal. The size of a numeric literal in standard data format characters is equal to the number of digits specified by the user.

If a literal conforms to the rules for the formation of numeric literals, but is enclosed in quotation marks, it is a nonnumeric literal.

IBM Extension

Boolean Literals: A Boolean literal is a character-string delimited on the left by the separator B" (or B') and on the right by the quotation mark separator. The character-string consists only of the character 0 or 1. The value of a Boolean literal is the character itself, excluding the delimiting separators. All Boolean literals are of the category Boolean.

End of IBM Extension

COBOL Words

A COBOL word can be:

- A user-defined word
- A system name
- A reserved word.

Any given COBOL word must be capable of being made unique. It can belong to only one of these classes.

You can not create a user-defined name that is the same as a system name or a reserved word; also, system names should all be different from reserved words. The maximum length of a COBOL word is 30 characters.

User-Defined Words: A user-defined word is a COBOL word that you create. Valid characters in a user-defined word are:

- A through Z
- 0 through 9
- - (hyphen).

The hyphen cannot appear as the first or last character in a user-defined word.

The convention used to represent user-defined words in this manual is described in "Format Notation" on page 14.

Table 3 shows a list of user-defined words, and gives the rules for forming them. The function of each user-defined word depends on how it is used in a particular clause or statement.

Table 3. User-Defined Words and Rules for Formation

User-Defined Words	**Rules for Formation**
Alphabet-Name Condition-Name Data-Name Record-Name File-Name Index-Name Mnemonic-Name Routine-Name Library-Name Text-Name Program-Name	Must contain at least one alphabetic character. Within each set, the name must be unique either because no other word is made up of an identical character-string, or because it can be made unique through qualification. (See "Methods of Referencing Data and Procedures" on page 15.)
Paragraph-Name Section-Name	Need not contain an alphabetic character. Other rules are the same as for the first set (above).
Level-Numbers: 01-49, 66, 77, 88	Must be a 1- or 2-digit integer. Need not be unique.
Segment-Numbers: 00-99	Must be a 1- or 2-digit integer. Need not be unique.

System-Names: A system-name is an IBM-defined name that is used to communicate with the system. A system-name can be:

- A computer-name
- A language-name
- An implementor-name
  - A function-name
  - An assignment-name
  - A user-name.

The function of each system-name is described with the format in which it appears; each is defined in the glossary.

Reserved Words: A reserved word is a COBOL word with a fixed meaning(s) or function. You can use reserved words only as specified in the formats for a COBOL source program.

A complete list of COBOL reserved words is given in Appendix G, "COBOL Reserved Words."

There are six types of reserved words:

- Keywords
- Optional words
- Connectives
- Special registers
- Special-character words
- Figurative constants.

The convention used to represent reserved words in this manual is described in "Format Notation" on page 14.

Keywords are words that are required within a given clause, entry, or statement. There are three types of keywords:

- Verbs, such as `ADD`, `READ`, `WRITE`
- Required words, which appear in clause, entry, or statement formats, such as the word `USING` in the `MERGE` statement
- Words with a specific functional meaning, such as `NEGATIVE` or `SECTION`.

The convention used to represent keywords in this manual is described in "Format Notation" on page 14.

Optional words are words that you can include in a clause, entry or statement. When you omit an optional word, the meaning of the COBOL program is unchanged.

The conventions used to represent optional words in this manual are described in "Format Notation" on page 14.

Connectives are words or punctuation characters used to combine parts of COBOL statements. There are three types of connectives:

- Qualifier connectives (`OF`, `IN`) associate a data name, condition name, text name, or paragraph name with its qualifier.
- Series connectives (the comma and semicolon) optionally link two or more consecutive operands in a series. (An operand is a data item or literal that is acted upon by the COBOL program.)
- Logical connectives (`AND`, `OR`, `AND NOT`, `OR NOT`) are used in forming a conditional expression.

Special registers are compiler-generated storage areas used primarily to store information produced through one of the specific COBOL features. Each such storage area has a fixed name and need not be further defined within the program. These special registers include:

IBM Extension

- DB-FORMAT-NAME

  (See "Common Input/Output Phrases" on page 370 under "Input/Output Statements" on page 370.)

End of IBM Extension

- DEBUG-ITEM (see "DEBUGGING FEATURES" on page 517).
- LINAGE-COUNTER (see "LINAGE-COUNTER Special Register" on page 308).
- DATE, DAY, TIME (see "ACCEPT Statement" on page 373).

Special-character words are arithmetic operators (+ – * / **) or relation characters (< > =). Arithmetic operators are described under "Arithmetic Expressions" on page 352. Relation characters are described in the relation condition description under "Conditional Expressions" on page 354.

Figurative constants name and refer to specific constant values:

- ZERO, ZEROES, or ZEROS represents the value 0 or one or more occurrences of the character 0, depending on context. ZERO can be numeric or nonnumeric, depending on context. For example, ZERO is considered to be nonnumeric when it is used in a relational expression in which it is compared with an alphanumeric data item.
- SPACE or SPACES represents one or more blanks or spaces. SPACE must be nonnumeric.
- HIGH-VALUE or HIGH-VALUES represents one or more occurrences of the character with the highest value in the collating sequence used. For the EBCDIC collating sequence in System/38-Compatible COBOL, the character is hex FF; for other collating sequences, the character used depends on the collating sequence. In a COBOL program, HIGH-VALUE is treated as a nonnumeric literal.
- LOW-VALUE or LOW-VALUES represents one or more occurrences of the character with the lowest value in the collating sequence used. For the EBCDIC collating sequence in System/38-Compatible COBOL, the character is hex 00; for other collating sequences, the character used depends on the collating sequence. In a COBOL program, LOW-VALUE is treated as a nonnumeric literal.
- QUOTE or QUOTES represents one or more occurrences of the quotation mark character and must be nonnumeric. The word QUOTE or QUOTES cannot be used in place of an apostrophe or a quotation mark to enclose a nonnumeric literal.

IBM Extension

When APOST is specified as a compiler option, the figurative constant QUOTE has the EBCDIC value of an apostrophe.

End of IBM Extension

- ALL literal: Represents one or more occurrences of the string of characters composing the literal and is nonnumeric. The literal must be either a nonnumeric literal, Boolean literal, or a figurative constant other than the ALL literal.

When a figurative constant is used, the word ALL is redundant and is used for readability only. The figurative constant, ALL literal, cannot be used with the DISPLAY, INSPECT, STRING, STOP, or UNSTRING statements.

The singular and plural forms of a figurative constant are equivalent, and you can use them interchangeably. For example, if DATA-NAME-1 is a 5-character data item, either of the following statements will fill DATA-NAME-1 with 5 spaces:

```
MOVE SPACE TO DATA-NAME-1.
MOVE SPACES TO DATA-NAME-1.
```

In any format, you can substitute a figurative constant for a nonnumeric literal; however, only the figurative constant ZERO (or ZEROS, or ZEROES) can be substituted for a numeric literal.

IBM Extension

The figurative constant ZERO can be used as a Boolean literal.

End of IBM Extension

The length of a figurative constant depends on the context of the program. The following rules apply:

- When a figurative constant is associated with a data item, the length of the figurative constant character string is equal to the length of the associated data item. This rule applies, for example, when you move a figurative constant to or compare a figurative constant with another item.
- When a figurative constant is not associated with another data item, the length of the character string is one character. This rule applies, for example, in the DISPLAY, INSPECT, STRING, STOP, and UNSTRING statements.

PICTURE Character Strings

A PICTURE character string consists of COBOL characters used as symbols in the PICTURE clause. The symbols you choose:

- Determine if the user-defined name associated with the PICTURE character string is:
  - Boolean
  - Numeric
  - Alphabetic
  - Alphanumeric.
- Define edited output fields.

Comments

A comment is a character string containing any combination of characters from the EBCDIC set. A comment serves only to document the program to make the program code more understandable to anyone who might be reading it. Comments take two forms:

- A comment entry in the Identification Division: For a further description of a comment entry, see “IDENTIFICATION DIVISION” on page 267.

- A comment line (preceded by an asterisk or a slash in Column 7) in any division of the program: For a further description of a comment line, see “Standard COBOL Format Used When Entering Code” on page 24.

Separators

A separator is a string of one or more punctuation characters or B" when used to delimit a Boolean literal. A separator can be placed next to another separator, or next to a character string. The characters that can be used as separators are shown in Table 4.

Table 4. Punctuation Characters

Punctuation Character	Meaning
	Space
,	Comma
;	Semicolon
.	Period
(	Left parenthesis
)	Right parenthesis
'	Apostrophe
"	Quotation mark
==	Pseudo text delimiter
B" or B‘	Opening delimiter for Boolean literal

The following rules apply to the formation of separators:

- A space is always a separator except when the space appears within a nonnumeric literal. When contained between the opening and closing quotation marks of a nonnumeric literal, the space is considered part of the literal. Wherever a space is used as a separator, more than one space can be used.
- A comma, semicolon, or period immediately followed by a space is a separator. These separators can appear only where explicitly allowed by COBOL rules.
- The left and right parentheses are separators. Parentheses must appear as balanced pairs of left and right parentheses to delimit subscripts, indexes, arithmetic expressions, or conditions.
- The apostrophe or quotation mark is a separator. An opening quotation mark must be immediately preceded by a space or a left parenthesis. A closing quotation mark must be immediately followed by one of the following separators: space, comma, semicolon, period, or right parenthesis. Quotation marks must appear as balanced pairs delimiting nonnumeric literals except when the literal is continued.
- The pseudo-text delimiter is a separator. An opening pseudo-text delimiter must be immediately preceded by a space. A closing pseudo-text delimiter must be immediately followed by one of the following separators: space, comma, semicolon, or period. Pseudo-text delimiters must appear as balanced pairs delimiting pseudo-text.

IBM Extension

- B" or B‘ is a separator when it is used to describe a Boolean literal. The B must immediately precede the quotation mark.

End of IBM Extension

Format Notation

In COBOL, basic formats are prescribed for the various elements of the language. In this manual, these formats are presented in a uniform system of notation that is explained in the following paragraphs. This notation is designed to assist the programmer in writing COBOL source statements.

- Reserved words are printed entirely in CAPITAL LETTERS. These words have preassigned meanings in COBOL. If any reserved word is misspelled, it is not recognized as a reserved word and can cause an error in the program. The two types of reserved words are keywords and optional words.
  - Keywords are required by the syntax of the format unless the portion of the format containing them is optional. In formats, keywords are shown in UNDERLINED CAPITAL LETTERS. A missing keyword is considered an error in the program.
  - Optional words are included only for readability. They can be included or omitted without changing the syntax of the program. Optional words are CAPITALIZED but not underlined.
- Words printed in lowercase letters represent information to be supplied by the user. All such words are defined in the text of this manual.
- For easier reference, some user-defined words are followed by a hyphen and a digit or letter. This suffix does not change the syntactical definition of the word.
- Braces ({ }) enclosing listed items indicate (1) that exactly one of the enclosed stacked items must be specified, and/or (2) when followed by an ellipsis, that the enclosed unit or item must be specified at least once.
- Square brackets ([ ]) indicate that the enclosed item or unit can be used or omitted, as requested for the program. When two or more items are stacked within brackets, one or none of them can be specified. When followed by an ellipsis, the item or unit can be repeated.
- The ellipsis (...) indicates that the immediately preceding unit can occur once or any number of times in succession. A unit can be a single lowercase word or a group of lowercase words and one or more reserved words enclosed in brackets and/or braces. When repetition is used, everything enclosed within the immediately preceding brackets or braces must be repeated.
- The arithmetic and logical operators (+, –, <, >, =) that appear in formats are required although they are not underlined.
- All punctuation and other special characters appearing in formats (except braces, brackets, ellipsis, commas, and semicolons) are required by the syntax of the format when they are shown; if they are omitted, an error occurs in the program. Additional punctuation can be specified, according to the punctuation rules given later in this manual.

- The required clauses and (when written) optional clauses must be written in the sequence shown in the format except where the accompanying text states otherwise.
- Comments, restrictions, and clarifications on the use and meaning of every format are contained in the description following the format.
-

```
IBM extensions to American National Standard, COBOL, X3.23-1974,
that are part of a command syntax are boxed like this sentence.
```

-

```
*************************************************************************
* COBOL clauses and statements that are syntax-checked, but are treated  *
* as documentation by the System/38-Compatible COBOL compiler, are boxed *
* like this sentence.                                                    *
*************************************************************************
```

- IBM extensions

IBM Extension

IBM extensions to American National Standard (ANS) COBOL, X3.23-1974, that are part of the text description begin with the paragraph heading, **IBM Extension** and are separated from the regular text as is this paragraph. Note that Chapter 5, “Interactive Processing Considerations and Example Programs” consists only of IBM extensions; the entire chapter is boxed like this paragraph.

End of IBM Extension

Methods of Referencing Data and Procedures

Every user-specified name defining an element in a COBOL program must be unique, either by having no other name with a character-string of the same value, or by making it unique through qualification, subscripting, or indexing. In addition, references to data and procedures can be either explicit or implicit. The rules for qualification and for explicit and implicit references follow.

Qualification

If the user-specified name you select is nested within a hierarchy of names, you can use the same name several times as long as its position within the hierarchy can be precisely pinpointed. This is done through qualification. Qualification of a name is the addition, to the name, of all the other names in the hierarchy that make it possible to precisely pinpoint the name.

To qualify a name, you would place one or more qualifying phrases after the name. For example: to qualify a name in the Data Division, you could use Format 1; to qualify a name in the Procedure Division you could use Format 2; and to qualify a name from a COPY library, you could use Format 3.

Format 1

```
{ data-name-1    } [ { OF } ]
{ condition-name } [ { IN } data-name-2 ] . . .
```

Format 2

```
paragraph-name [ { OF } section-name ]
               [ { IN } ]
```

Format 3

```
text-name [ { OF } library-name ]
          [ { IN } ]
```

In Data Division references, all qualifying data-names must be associated with a level indicator or level-number. Therefore, two identical data-names must not appear as subordinate entries in a group item unless they can be made unique through qualification. Names associated with a level indicator (FD and SD) are the highest level in the hierarchy. Next highest are those associated with level-number 01. Names associated with level-numbers 02 through 49 are at successively lower levels in the hierarchy.

In the Procedure Division, two identical paragraph-names must not appear in the same section. A section-name is the highest and only qualifier available for a paragraph-name.

The following example illustrates the use of identical names in a section hierarchy:

```
01 FIELD-A
   02 FIELD-B
      05 SUB1
         07 SUB2
   02 FIELD-C
         07 SUB1
```

A hierarchy includes all subordinate entries to the next equal or higher level-number. Therefore, in the above example all entries are in the hierarchy of FIELD-A. All entries from FIELD-B to, but not including, FIELD-C are in the hierarchy of FIELD-B.

In the hierarchy of FIELD-A, SUB1 can be used twice; once as subordinate to FIELD-B and once as subordinate to FIELD-C. In references to SUB-1, it must be qualified as SUB-1 OF FIELD-B or SUB-1 OF FIELD-C. Within FIELD-B or FIELD-C, SUB1 cannot be subordinate to itself.

In any hierarchy, the name associated with the highest level must be unique and cannot be qualified.

No matter what qualification is available, no name can be both a data-name and a procedure-name.

Enough qualification must be specified to make the name unique; however, it may not be necessary to specify all the levels of the hierarchy. For example, if more than one file has records that contain the field `EMPLOYEE-NO` but only one of the files has a record named `MASTER-RECORD`, then specifying `EMPLOYEE-NO OF MASTER-RECORD` sufficiently qualifies `EMPLOYEE-NO`. `EMPLOYEE-NO OF MASTER-RECORD OF MASTER-FILE` is valid but unnecessary.

Qualification Rules

The following rules for qualification apply:

- Each qualifier must be of a successively higher level and must be within the same hierarchy as the name it qualifies.
- The same name must not appear at two levels in a hierarchy unless it can be qualified.
- If a data-name or condition-name is assigned to more than one data item, the data item must be qualified each time it is referenced, with this exception: in the `REDEFINES` clause, qualification is unnecessary and must not be used.
- A paragraph-name must not be duplicated within a section. When a paragraph-name is qualified by a section-name, the word `SECTION` must not appear. A paragraph-name need not be qualified when referred to within the section in which it appears.
- Library-name must be unique in the system. Therefore, the first 10 characters of library-name must be unique.
- Text-name (member-name) must be qualified by the file-name and library-name in which it resides. If no library is specified, the library list `*LIBL` is searched.

IBM Extension

File-name is optional for the `COPY` statement, format 1. If file-name is not specified, the default is `QCBLSRC`.

End of IBM Extension

- When a data-name is being used as a qualifier, it cannot be subscripted.
- A name can be qualified even when it does not need qualification.
- If more than one combination of qualifiers ensures uniqueness, then any of these combinations can be used.
- Duplicate section-names are not allowed.
- A data-name cannot be the same as a section-name or a paragraph-name.
- A section-name cannot be the same as a paragraph-name.
- If a data-name cannot be made unique by qualification, duplication of this data-name is not allowed.
- The complete list of qualifiers for one data-name must not be the same as a partial list of qualifiers for another data-name.
- A maximum of 48 qualifiers (49 qualifiers for file data) can be specified.

- LINAGE-COUNTER must be qualified each time it is referenced if more than one file description entry containing a LINAGE clause has been specified in the source program.

Subscripting and Indexing

Subscripts and indexes can be used only when reference is made to an individual element within a table of elements that have not been assigned individual data-names. Subscripting and indexing are explained under "TABLE HANDLING" in Chapter 11, "Using the Additional COBOL Functions."

Identifier

An identifier is a term used to reflect that a data-name, if not unique in a program, must be followed by some syntactically correct combination of qualifiers, subscripts, or indexes sufficient to ensure uniqueness. The general formats for identifiers are as follows:

Format 1

```
data-name-1 [ { OF } data-name-2 ] ... [ ( subscript-1 [ , subscript-2 [ , subscript-3 ] ] ) ]
            [ { IN }                 ]   [                                                  ]
```

Format 2

```
data-name-1 [ { OF } data-name-2 ] ... [ ( { index-name-1 [ { } literal-2 ] } )
            [ { IN }                 ]   [   { literal-1                     }   ]

    [   { index-name-2  [ { } literal-4 ]}   [   { index-name-3  [ { } literal-6 ] } ]   ]
    [ , { literal-3                     }   ]   [ , { literal-5                     } ]   ]
```

Restrictions on qualification, subscripting, and indexing follow:

- A data-name must not be subscripted or indexed when that data-name is being used as an index, subscript or qualifier.
- Indexing is not permitted when subscripting is not permitted.
- An index can be modified only by the SET, SEARCH, and PERFORM statements. Data items described by the USAGE IS INDEX clause permit the values associated with index-names to be stored as a binary occurrence number. Such data items are called index data items.
- In the above format literal-1, literal-3, literal-5 must be positive numeric integers and literal-2, literal-4, literal-6 must be unsigned numeric integers.

Condition-Name

A condition-name is a user-defined word that is assigned a specific value or range of values. A condition-name can alternatively be a user-defined word that is assigned the status of an IBM-defined switch or device.

Each condition-name must be unique, or it must be made unique through qualification, and/or indexing, or subscripting.

If qualification is used to make a condition-name unique, the associated conditional variable may be used as the first qualifier. If qualification is used, the hierarchy of names associated with the conditional variable or the conditional variable itself must be used to make the condition-name unique.

If references to a conditional variable require indexing or subscripting, then references to any of its condition-names also require the same combination of indexing or subscripting.

The format and restrictions on the combined use of qualification, subscripting, and indexing of condition-names are the same as that for identifiers except that `data-name-1` is replaced by `condition-name-1`.

In the general formats, condition-name refers to a condition-name that is qualified, indexed, or subscripted as necessary.

Explicit and Implicit References

COBOL source program references can be either explicit or implicit in three instances: data attribute specification, Procedure Division data references, and transfers of control.

Data Attribute Specification

Explicit attributes are specified in COBOL coding. If a data attribute is not specified in COBOL coding, it takes on a default value. These default values are implicit attributes.

For example, the `ACCESS MODE` clause in the file-control entry need not be specified. If the clause is omitted, the compiler provides the default value, `ACCESS MODE IS SEQUENTIAL`. This clause is then an implicit attribute. If this same attribute, `ACCESS MODE IS SEQUENTIAL`, is specified in the COBOL coding, it is an explicit attribute.

Procedure Division Data References

Procedure Division statements can refer to data items either explicitly or implicitly.

An explicit reference occurs when the data-name of the item is written in a COBOL statement or when the data-name is copied into the program through a `COPY` statement. An implicit reference occurs when the data-name is referred to by a COBOL statement without the name being written in that statement.

For example, when `USE AFTER STANDARD EXCEPTION/ERROR PROCEDURE ON INPUT` is specified, an implicit reference is made to each file-name that identifies an input file. See “EXCEPTION/ERROR Declarative” in Chapter 10, “Procedure Division” for a further description.

Transfers of Control

In the Procedure Division, program flow transfers control from statement to statement in the order they are written unless an explicit control transfer is specified or no next executable statement exists. (See the note below.) This normal program flow is an implicit transfer of control.

In addition to the implicit transfers of control between consecutive statements, implicit transfer of control also occurs when the normal flow is altered without the processing of a procedure branching statement. COBOL provides implicit transfers of control that override the statement-to-statement transfers of control under the following conditions:

- After processing the last statement of a procedure being run under control of another COBOL statement. COBOL statements that control the running of a procedure are MERGE, PERFORM, SORT, and USE.
- During the processing of a SORT or MERGE statement when control is transferred to any input or output procedure.
- During the processing of any COBOL statement that causes the calling of a Declarative procedure.
- At the end of any Declarative procedure.

COBOL also provides explicit transfers of control through the running of a procedure branching or a conditional statement. Lists of procedure branching and conditional statements are given under “Procedure Division Organization” in Chapter 10, “Procedure Division.”

Note: The term *next executable statement* refers to the next COBOL statement to which control is transferred according to the rules given above. No next executable statement can follow:

- The last statement in a Declarative procedure that is not being run under control of another COBOL statement.
- The last statement in a COBOL program when the paragraph in which it appears is not being processed under control of another COBOL statement.

Chapter 2. Coding and Entering Your Program

System/38 Environment

Since many programs have been written and are meant to run on a System/38, the AS/400 system has been designed with the ability to emulate a System/38. By entering the CL command CALL QCL, you access a "System/38 environment". When in this environment you can create, compile, run, and process System/38 programs.

Notes:

1. This manual is written from the viewpoint that you are working with System/38-Compatible COBOL from within the System/38 environment.
2. CL commands referred to in this manual are CL commands used in the System/38 environment.

To exit from the System/38 environment back into the AS/400 system environment, you use the F3 key.

Entering the Source Program into the System

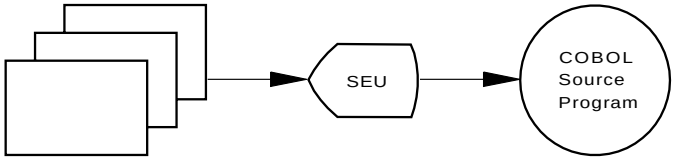
After you have written your COBOL program on COBOL coding forms (the COBOL coding form is shown in Figure 3 on page 25), you must enter it into source files in the system. You can also enter the code directly into the system without first manually completing the coding forms.

These source files should have a record length of 92. However, the System/38-Compatible COBOL compiler also supports a record length of 102. In addition to the usual fields of sequence number (6 characters), last modified date (6 characters), and the data (80 characters), a field of 10 characters that can contain additional information is placed at the end of the record (positions 93-102). This information is not used by the COBOL compiler, but is placed on the extreme right of the compiler listing. You are responsible for placing information into this field. If you want to use the additional field, create a source file with a record length of 102.

The source file QCBLSRC has a record length of 92.

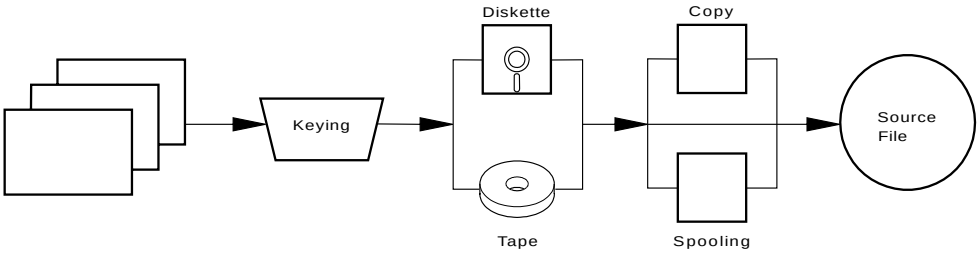
The normal ways of entering a source program are:

- You can enter it interactively by using the Source Entry Utility (SEU) and the System/38-Compatible COBOL syntax checker.



  To call SEU, enter the CL command STRSEU (Start SEU). You can also use the CL Command EDTSRC (Edit Source) in the System/38 environment with TYPE(*CBL). SEU internally treats the type as (CBL38). Refer to the *SEU* for further information on this editing facility.

- You can enter your source program in a batch manner (for example, from diskettes) by using the OS/400 copy function:



Refer to the *Data Management Guide* for more information on how to use the copy or spooling function for batch entry of the source program.

Using SEU and the System/38-Compatible COBOL Syntax Checker to Enter Source Programs

SEU provides special display screen formats for COBOL that correspond to the COBOL coding form to help you enter your COBOL source programs. If you specify the TYPE(CBL38) parameter on the STRSEU command (or in the System/38 environment, specify TYPE(*CBL) on the EDTSRC command), SEU calls a COBOL syntax checker that checks each line for errors as you enter it. Figure 2 on page 23 shows a display screen format, the relationship between the headings on the COBOL Coding Form and the labels on the display screen, and where you can enter the source on the display screen.

The COBOL syntax checker finds invalid entries in the source statements and displays error messages that allow you to correct the errors before compiling the program. Since the COBOL syntax checker checks only one statement at a time, independently of statements that precede or follow it, only syntax errors can be detected. No interrelational errors, such as undefined names and invalid references to names, are detected. These errors are detected by the COBOL compiler when the program is compiled.

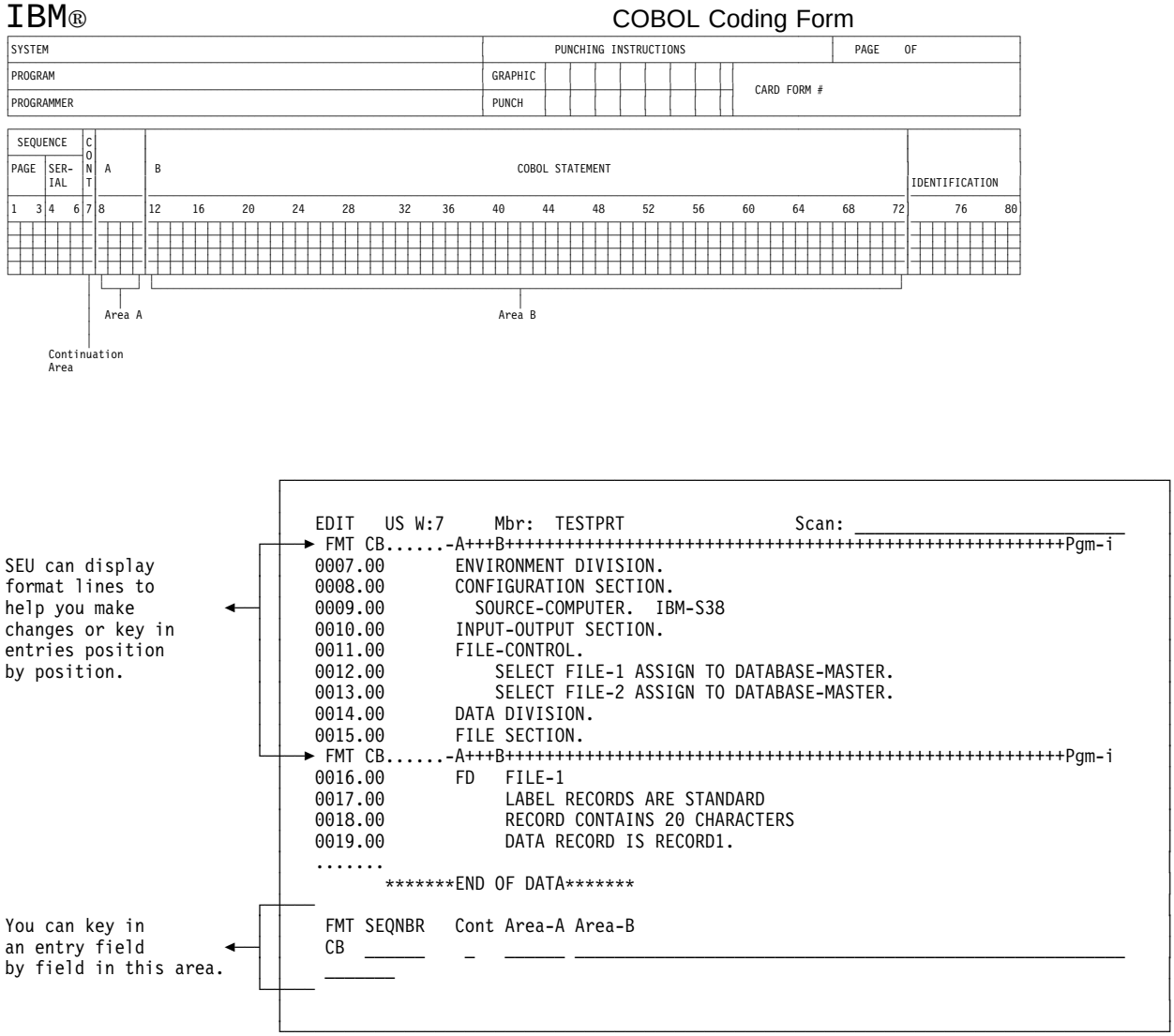


Figure 2. Relationship between COBOL Coding Form and an SEU Display

Syntax checking for COBOL source functions is subject to the following rules.

- Source code on a line with an asterisk (*) or a slash (/) in column 7 is not syntax checked. An asterisk indicates a comment line, a slash indicates a comment line and page eject.
- Syntax checking occurs line by line as source code is entered. The error messages that are generated by lines consisting of incomplete statements disappear when the statements are completed. In the example:

  ```
  ADD A
  TO BCD.
  ```

  an error message is generated after the first line is entered and disappears after the second line is entered, when the statement is completed.
- Any time a source line is entered or changed, up to 45 lines of source may be syntax checked as one unit. The length of a single unit of syntax checking is determined by extending from an entered or changed line as follows:

  - A unit of syntax checking extends towards the beginning of the source member until the first source line, or a line that ends in a period is encountered.
  - A unit of syntax checking extends towards the end of the source member until the last source line, or a line that ends in a period is encountered.
  - If this unit spans more than 45 source lines (not including comment lines), the system responds with a message.
- If there is an error in a unit of syntax checking, the entire unit is presented in reverse image. The message at the bottom of the display screen refers to the first error in the unit.
- No compiler options are honored to control syntax checking.

  For example, the syntax checker accepts either quotes or apostrophes as non-numeric delimiters as long as they are not mixed within one unit of syntax checking. It does not check whether the delimiter is the one specified in the CRTCBLPGM command or in the PROCESS statement.
- The syntax checker requires the first SEU clause of the following paragraphs to be entered on the same line as the paragraph name:

  ```
  SPECIAL-NAMES.
  PROGRAM-ID.
  AUTHOR.
  INSTALLATION.
  DATE-WRITTEN.
  DATE-COMPILED.
  SECURITY.
  SOURCE-COMPUTER.
  OBJECT-COMPUTER.
  ```

  This is not a requirement for compiling.
- Character replacement specified by the CURRENCY and DECIMAL-POINT clauses of the SPECIAL-NAMES paragraph is not honored during interactive syntax checking.

For a complete description of how to enter a source program using SEU, refer to the *SEU*.

Standard COBOL Format Used When Entering Code

COBOL programs must be written in the standard COBOL format, as a series of 80-character lines. The output listing of the source program is printed in this same format. The COBOL coding form for each line is shown in Figure 3 on page 25.

Sequence Numbers (Columns 1-6)

Sequence numbers are written in columns 1 through 6. A sequence number numerically identifies each line to be compiled by the COBOL compiler. Sequence numbers are optional. A sequence number, if used, must consist of six digits in the sequence number area, (including the preprinted digits in columns 4 and 5).

If sequence numbers are used in the source program, they must be in ascending order. If sequence numbers are out of sequence, the compiler accepts them in the order read and generates a warning message.

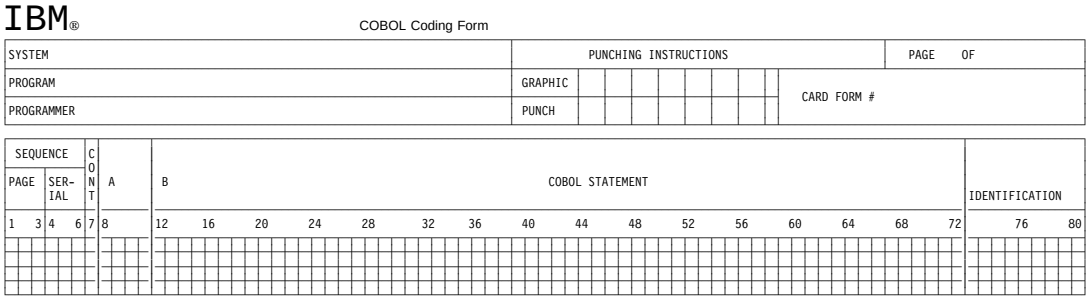
IBM Extension

The user can suppress sequence checking at compile time by specifying NOSEQUENCE.

If the NUMBER option is specified, the sequence numbers from columns 1 through 6 are used; otherwise the source sequence numbers provided in the source file are used.

If the LINENUMBER option is specified, the compiler-generated sequence numbers are used for reference numbers. The option combines program source code and source code introduced by COPY statements into one consecutively numbered sequence. Use this option if you specify FIPS flagging.

End of IBM Extension



Columns 1-6 represent the sequence number area.
Column 7 is the continuation area.
Columns 8-11 represent Area A } Used for writing COBOL source statements.
Columns 12-72 represent Area B }
Columns 73-80 are used to identify the program.

Figure 3. IBM COBOL Coding Form and Standard COBOL Format

Continuation Area (Column 7)

The continuation area is used to indicate the continuation of words and nonnumeric literals from the previous line onto the current line, to specify debugging lines, or to indicate that the text on this line is to be treated as a comment.

Area A (Columns 8-11) and Area B (Columns 12-72)

COBOL elements that can begin in Area A and specific COBOL elements that can follow them are shown in Table 5.

The basic skeleton of a COBOL program is shown in Figure 4 on page 27.

Table 5 (Page 1 of 2). Sequence of Elements in Area A and Area B

Elements That Must Begin in Area A	Must Be Followed Immediately By	Placement of Following Elements
Division header	(In Procedure Division) USING phrase	Same or next line (Area B)
	Section header, paragraph header, paragraph-name, or (in Procedure Division) keyword DECLARATIVES	Next line (Area A)

Table 5 (Page 2 of 2). Sequence of Elements in Area A and Area B

Elements That Must Begin in Area A	Must Be Followed Immediately By	Placement of Following Elements
Section header	(In Declaratives section) `USE` statement	Same or next line (Area B)
	Paragraph header, paragraph-name, (after `USE`, if specified), level indicator, or level-number	Next line (Area A)
Paragraph header or paragraph-name	Identification Division entry, Environment Division entry, or Procedure Division sentence	Same or next line (Area B)
Level indicator, level-numbers 01 and 77	Data-name	Same or next line (Area B)
Keyword `DECLARATIVES`	Declaratives section-name	Next line (Area A)
Keywords `END DECLARATIVES`	Section header	Next line (Area A)

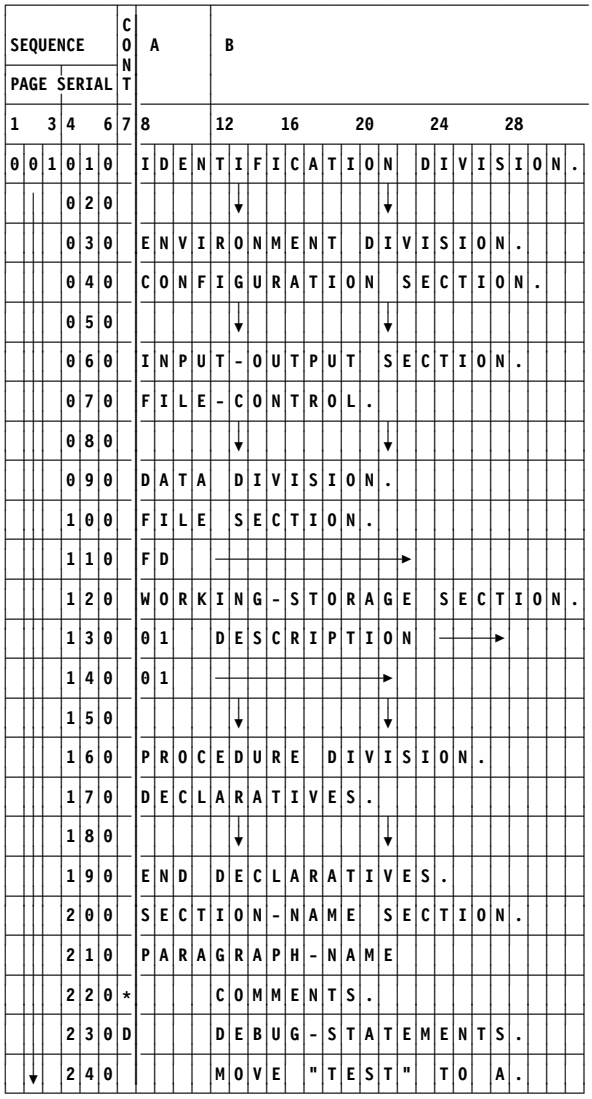


Figure 4. Basic Skeleton of a COBOL Program

Special Considerations

Some lines in a COBOL program require additional rules. A discussion of each follows.

Division Header

A division header must be immediately followed by a period except when a USING phrase is specified with a Procedure Division header. Except for the USING phrase, no text can appear on the same line.

Section Header

A section header must be immediately followed by a period except when Procedure Division segment numbers are specified. In the Environment and Procedure Divisions, a section consists of paragraphs. In the Data Division, a section consists of Data Division entries.

Paragraph Header, Paragraph-Name

In both the Identification Division and the Environment Division, a paragraph consists of a paragraph header followed by one or more entries in Area B of the coding form. An entry consists of one or more clauses. In the Procedure Division, a paragraph consists of a paragraph-name followed by one or more sentences in Area B. A sentence consists of one or more statements; a statement is a syntactically valid combination of a COBOL verb and its operands. Entries and sentences must be ended with a period followed by a space.

Successive entries or sentences begin in Area B. The entries are either on the same line as the last entry or sentence, or they are on the next succeeding nonblank noncomment line.

Data Division Entries

Each Data Division entry begins with a level indicator or level-number followed by a space. On the same line is a data-name in Area B, followed by a sequence of independent clauses describing the item. Each clause, except the last, is followed by a space (or optionally by a comma or semicolon and a space). The last clause in the entry must be ended with a period followed by a space.

Successive clauses begin in Area B. The clauses are either on the same line as the preceding clause, or on the next succeeding nonblank noncomment line.

A level indicator (`FD`, `SD`) must begin in Area A and be followed by a space. For a further description of level indicators, see "Data Division Organization" in Chapter 9, "Data Division."

A level-number is a 1- or 2-digit integer with one of the following values: 1 through 49, 66, 77, or 88. At least one space must follow the level-number.

Level-numbers 01 and 77 must begin in Area A. The associated record-name or item-name must appear in Area B. Level-numbers 02 through 49, 66, and 88 can begin in either Area A or Area B.

DECLARATIVES and END DECLARATIVES

In the Procedure Division, the keywords `DECLARATIVES` and `END DECLARATIVES` begin and end the Declaratives portion of the source program. Both of these keywords must begin in Area A and be followed immediately by a period. No other text can appear on the same line. After the keyword `END DECLARATIVES`, no text can appear before the following section header.

Program Spacing

In writing a COBOL program, rules for indentation, continued lines, comment lines, debugging lines, and blank lines must be observed.

Indentation

Within an entry or sentence, successive lines in Area B can have the same format or can be indented to clarify program logic. The output listing is indented only if the input statements are indented. Indentation does not affect the syntax of the program. The amount of indentation can be chosen by the user, subject only to the restrictions on the width of Area B.

Continuation of Lines

Any sentence, entry, clause, or phrase that requires more than one line can be continued in Area B of the next succeeding noncomment line. The line being continued is called the continued line; the succeeding lines are continuation lines. Area A of a continuation line must contain only spaces.

If there is no hyphen in the continuation area (Column 7) of a line, the last character of the preceding line is assumed to be followed by a space.

If there is a hyphen in the continuation area of a line, the first nonblank character of this continuation line immediately follows the last nonblank character of the continued line without any intervening space. However, this restriction does not apply to nonnumeric literals.

If the continued line contains a nonnumeric literal without a closing quotation mark, all spaces at the end of the continued line (through Column 72) are considered to be part of the literal. The continuation line must contain a hyphen in the continuation area, and the first nonblank character in Area B must be a quotation mark. The continuation of the literal begins with the character immediately following the quotation mark.

A pair of quotation marks indicating a single quotation mark in the value of the literal must occur on the same line. Likewise, both characters composing the separator == or B" must be on the same line.

Comment Lines

A comment line is any line with an asterisk or slash in the continuation area of the line. The comment may be written anywhere in Area A and Area B of that line. The comment may consist of any combination of characters from the EBCDIC set.

If an asterisk is placed in the continuation area, this comment line is printed in the output listing immediately following the last preceding line.

If the slash is placed in the continuation area, the current page of the output listing is ejected, and the comment line is printed on the first line of the next page.

The asterisk or slash and the comment are produced only on the output listing. They are treated as documentation by the compiler.

Successive comment lines are allowed. Each must begin with an asterisk or slash in the continuation area.

Comment lines are not allowed before the Identification Division header.

Debugging Lines

A debugging line is any line with a D coded in the continuation area. Rules for the formation of debugging lines are given under “DEBUGGING FEATURES” in Chapter 11, “Using the Additional COBOL Functions.”

Blank Lines

Blank lines contain nothing but spaces from Column 7 through Column 72. A blank line may appear anywhere in a program except immediately preceding a continuation line.

Overall Punctuation Rules

Any punctuation character included in a PICTURE character-string, a comment character-string, or a nonnumeric literal is not considered to be a punctuation character but rather is considered to be part of the character-string or literal.

A comma, period, or semicolon followed by a space in or at the end of a PICTURE character-string is a separator and terminates the PICTURE character-string. The comma and semicolon are used only for readability.

Punctuation rules for each division of the COBOL source program follow.

Identification Division

Commas and semicolons can be used in the comment-entries. The PROGRAM-ID paragraph must end with a period followed by a space.

Environment Division

Commas or semicolons can separate successive clauses and successive operands within clauses. The SOURCE-COMPUTER, OBJECT-COMPUTER, SPECIAL-NAMES, and I-O-CONTROL paragraphs must each end with a period followed by a space. In the FILE-CONTROL paragraph, each file-control entry must end with a period followed by a space.

Data Division

Commas or semicolons may separate successive clauses and operands within clauses. File (FD), Sort/Merge file (SD), and data description entries must each end with a period followed by a space.

Procedure Division

Commas or semicolons may separate successive statements within a sentence and successive operands within a statement. Each sentence and each procedure must end with a period followed by a space.

COPY Statement

Prewritten source program entries can be included in a source program at compile time. Thus, an installation can use standard file descriptions, record descriptions, or procedures without recoding them. These entries and procedures can be saved in files. They are included in the source program by means of the COPY statement which includes them as part of the source program at compile time.

Format 1

```
COPY text-name [ { OF } file name [ -library name ] ]
               [ { IN }                              ]

     [          {                                                  }       ]
     [          {   { ==pseudo-text-1== }    { ==pseudo-text-2== } }       ]
     [ REPLACING { , { identifier-1      } BY { identifier-2      } } . . . ] .
     [          {   { literal-1         }    { literal-2         } }       ]
     [          {   { word-1            }    { word-2            } }       ]
     [          {                                                  }       ]
```

Compilation of the source program containing COPY statements is logically equivalent to processing all COPY statements before processing the resulting source program.

The effect of processing a COPY statement is that the text associated with text-name is copied into the source program, logically replacing the entire COPY statement beginning with the word COPY and ending with the period, inclusive. When the REPLACING phrase is not specified, the text is copied unchanged.

The text-name is the name of the member to be copied. The text-name must begin with an alphabetic character. The first ten characters of the text-name are used as the member name; these first ten characters must, therefore, be unique within one file.

If text-name is not qualified, QCBLSRC is assumed as the file name. If the file name is not qualified by a library name, it is assumed to reside in a library in the library list, *LIBL.

The library name , file name , and text-name must follow the rules for formation of any AS/400 System/38 environment name.

A COPY statement can appear in the source program anywhere that a character-string or a separator can appear. However, a COPY statement must not be specified within the resulting copied text. Each COPY statement must be preceded by a space, and followed by a period and a space.

Comment lines can appear in copied text. Comment lines in text are copied into the source program unchanged and are interpreted logically as a single space.

Debugging lines can appear in copied text. When a COPY statement is specified on a debugging line, the copied text is treated as though it appeared on a debugging line except that comment lines in the text appear as comment lines in the resulting source program.

The syntactic correctness of the entire COBOL source program cannot be determined until all COPY statements have been completely processed, because the syntactic correctness of the copied text cannot be independently determined.

Text copied from a member is placed into the same area of the resultant program as it is in the member. Copied text must conform to the rules for standard COBOL format.

IBM Extension

Format 2 COPY statements are used to generate COBOL Data Division statements to describe files that exist on the system.

Note: The use of the term Format 2 COPY statement throughout this manual is intended as a reference to the COPY statement, DDS or DD format.

Format 2

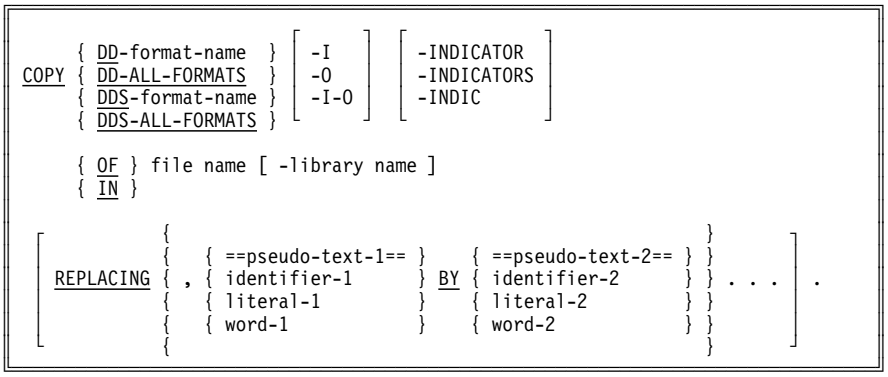


Figure 5. Format 2 COPY Statement

For more detailed information about the Format 2 COPY statement see “Externally Described/Program Described Files” on page 211.

End of IBM Extension

REPLACING Phrase

In the REPLACING phrase, each operand can consist of one of the following: pseudo-text, an identifier, a literal, or a COBOL word. When the REPLACING phrase is specified, each operand-1 from the copied text is replaced by its associated operand-2.

Pseudo-text is a sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters (==). Both characters of each pseudo-text delimiter must appear on one line; however, character-strings within pseudo-text can be continued.

Pseudo-text-1 must not be null; neither can it consist solely of the space character and/or of comment lines.

Pseudo-text-2 can be null. It can consist solely of space characters and/or comment lines.

Each identifier can be defined in any Data Division section.

Each literal can be numeric or nonnumeric.

Each COBOL word can be any single COBOL word.

Note: Sequences of code (such as file and data descriptions, error and exception routines, and so on) that are common to a number of programs can be cataloged

and used in conjunction with the COPY statement. If naming conventions are established for such common code, then the REPLACING phrase need not be specified. If the names will change from one program to another, then the REPLACING phrase can be used to supply meaningful names for this program.

REPLACING Phrase Processing

When the REPLACING phrase is specified, the text is copied, and each properly matched occurrence of operand-1 within the text is replaced by the associated operand-2.

For purposes of matching, each identifier-1, literal-1, or word-1 is treated as pseudo-text containing only identifier-1, literal-1, or word-1 respectively. Separator spaces in identifiers are optional in both the copied text and the comparison text.

The comparison proceeds as follows:

- Any separator comma, semicolon, and/or space preceding the leftmost word in the text is copied into the source program. Beginning with the leftmost text word and the first operand-1 specified in the REPLACING phrase, the entire REPLACING operand that precedes the keyword BY is compared to an equivalent number of contiguous text words.
- Operand-1 matches the text only if the ordered sequence of text words in operand-1 is equal, character for character, to the ordered sequence of words. For matching purposes, each occurrence of a comma or semicolon separator is considered to be a single space. However, when operand-1 consists solely of a separator comma or semicolon, it participates in the match as a text word. In this case, the space following the comma or semicolon separator can be omitted. Each sequence of one or more space separators is considered to be a single space.
- If no match occurs, the comparison is repeated with each successive operand-1 (if specified) until either a match is found or there are no further REPLACING operands.
- Whenever a match occurs between operand-1 and the copied text, the associated operand-2 is copied into the source program in the place of operand-1.

IBM Extension

Operand-2 is copied in the place of operand-1 unless pseudo-text-2 positioning rules cause the replacement to be inserted in a different area.

End of IBM Extension

- When all operands have been compared and no match is found, the leftmost text word is copied into the source program.
- The next successive uncopied text word is then considered the leftmost text word, and the comparison process is repeated, beginning with the first operand-1. The process continues until the rightmost copied text word has been compared.

- A comment line occurring in operand-1 and in the copied text is interpreted for matching purposes as a single space. A comment line appearing in operand-2 is copied unchanged into the source program.
- Debugging lines are not permitted in operand-1. Debugging lines, however, are permitted in copied text and in operand-2. Text words in a debugging line are matched as if no D appeared in column 7.
- Text words after replacement are placed in the source program according to standard COBOL format rules.

Notes:

1. Arithmetic and logical operators are considered to be text words and can be replaced only through the pseudo-text phrase.
2. When a figurative constant is operand-1, it will match only exactly as specified. For example, if ALL "AB" is specified in the copied text, then "ABAB" is not considered a match. Only ALL "AB" is considered a match.

COPY Statement Example

In this example, the member PAYREC consists of the following Data Division entries:

```
02 B PIC S99.
02 C PIC S9(5)V99.
02 D PIC S9999 OCCURS 1 TO 52 TIMES
     DEPENDING ON B OF A.
```

The user can use the COPY statement in the Data Division of a program as follows:

```
01 PAYROLL.  COPY PAYREC OF PAYFILE.
```

In this program, the member is then copied. The resulting entry is treated as if it had been written as follows:

```
01 PAYROLL.
   02 B PIC S99.
   02 C PIC S9(5)V99.
   02 D PIC S9999 OCCURS 1 TO 52 TIMES
        DEPENDING ON B OF A.
```

To change some (or all) of the names within the member, the user can use the REPLACING phrase:

```
01 PAYROLL. COPY PAYREC OF PAYFILE
   REPLACING A BY PAYROLL
               B BY PAY-CODE
               C BY GROSS-PAY.
```

In this program, the member is copied. The resulting entry is treated as if it had been written as follows:

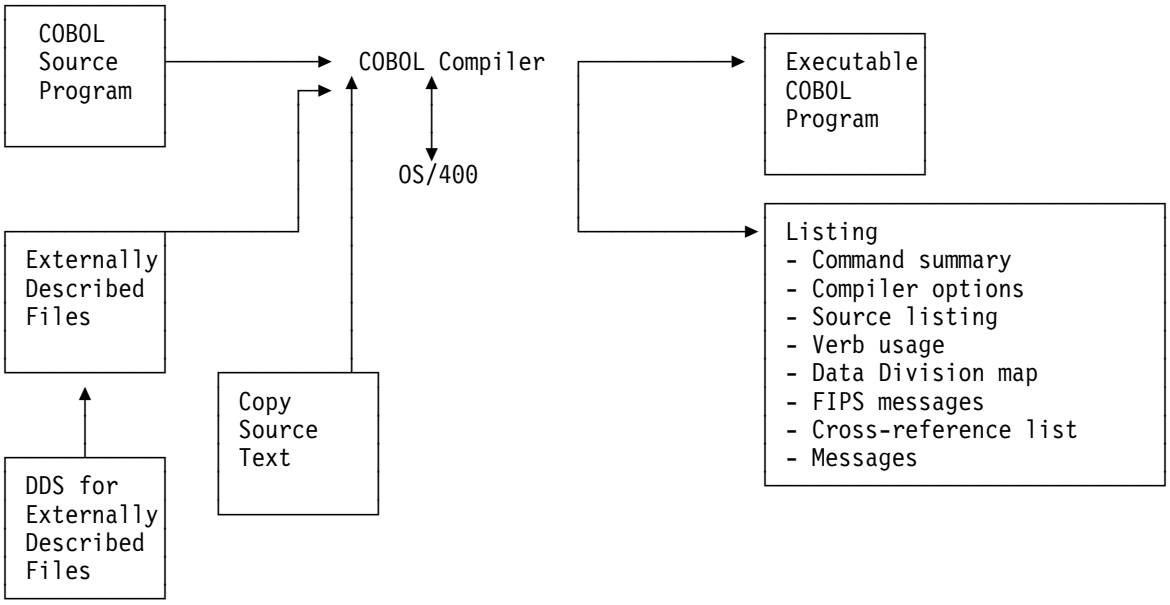
```
01 PAYROLL.
   02 PAY-CODE     PIC S99.
   02 GROSS-PAY    PIC S9(5)V99.
   02 D            PIC S9999 OCCURS 1 TO 52
                     TIMES DEPENDING ON
                     PAY-CODE OF PAYROLL.
```

The changes shown are made only for this program. The entry as it appears in the member remains unchanged.

Chapter 3. Compiling Your Program

After you have entered the source program into the system, you need to compile it to produce an object program that can be run.

To compile a System/38-Compatible COBOL program, you must use the CL command CRTCBLPGM (Create COBOL Program) in the System/38 environment. If the Format 2 COPY statement is used in the program to access externally described files, the Operating System/400* (OS/400*) provides information about the externally described files to the compiled program. The result of the compilation is a COBOL object program and a listing:



During compilation, the compiler syntax checks the COBOL source program line by line as well as checking the interrelationships between the lines.

Compiler Options

You can specify various compiler options either by using the OPTION parameter of the CRTCBLPGM command or from within the program by using the PROCESS statement. Any options specified in the PROCESS statement override the corresponding options on the CRTCBLPGM command. The PROCESS statement is discussed later in this chapter.

Create COBOL Program Command

To compile a System/38-Compatible COBOL source program into an object program, you must enter the CRTCBLPGM (Create COBOL Program) command. This calls the System/38-Compatible COBOL compiler. The command is valid in batch and interactive jobs, or from other programs in the System/38 environment. You need QTEMP on the library list, but QCBL is no longer required.

Note: CRTCBLPGM is used to create System/38-Compatible COBOL programs in the System/38 environment.

If the COBOL compiler terminates, the escape message CBL9001 is issued. A CL program can monitor for this exception by using the CL command MONMSG (Monitor Message).

All object names specified and entered with the CRTCBLPGM command must be composed of alphanumeric characters, the first of which must be alphabetic. The length of the names cannot exceed 10 characters. See the *CL Reference* for a detailed description of object naming rules and for a complete description of the CL command syntax.

When the CRTCBLPGM command is issued in a CL program, concatenation expressions can be used for all parameter values. See the *CL Reference* for more information about concatenation expressions.

Note: The number of entries in the Object Definition Table (ODT) and the amount of storage required by a COBOL program varies with the number and kinds of COBOL statements used in the program. A combination of these factors can cause the AS/400 system internal size limits for the program to be exceeded. If this occurs, use the *NOUNREF option of the GENOPT parameter. If the problem persists, the program must be rewritten.

When the *NOUNREF option is specified, only names that are referenced or are needed for data structuring are defined. This option is useful when the program contains many unreferenced identifiers.

If member type is not CBL38, a warning will be issued.

Completing the CRTCBLPGM Prompt Screens

If you key in the CRTCBLPGM command and press the F4 key, the following screen is shown on the display (see Figure 6 on page 39).

```
CRTCBLPGM                  Create COBOL Program

Type choices, press Enter.

Program  . . . . . . . . . . .   *PGMID____     Name, *PGMID
  Library  . . . . . . . . . .     QGPL______   Name
Source file  . . . . . . . . .   QCBLSRC___     Name
  Library  . . . . . . . . . .     *LIBL_____   Name, *LIBL
Source member  . . . . . . . .   *PGM______     Name, *PGM
Source listing options . . . .   __________     *SOURCE, *NOSOURCE, *SRC...
              + for more values  __________
Generation options . . . . . .   __________     *NOLIST, *LIST, *NOXREF...
              + for more values  __________
Generation severity level  . . .  29____       0-29
Print file . . . . . . . . . .   QSYSPRT___     Name
  Library  . . . . . . . . . .     *LIBL_____   Name, *LIBL
FIPS flagging level  . . . . .   *NO            *NO, *L, *LI, *HI, *H
Flagging severity  . . . . . .   0____          0-99
User profile . . . . . . . . .   *USER          *USER, *OWNER
Public authority . . . . . . .   *NORMAL        *NORMAL, *ALL, *NONE
                                                                    More...
F3=Exit   F4=List   F5=Refresh   F11=Keywords   F12=Previous   F13=Prompter help
```

Figure 6. The First CRTCBLPGM Prompt Screen

Each parameter on the screen displays a default value. Move the cursor past items where you want the default value to apply. Type over any items where you want to set a different value or option. Scroll Up to display the next screen of additional parameters.

```
CRTCBLPGM                  Create COBOL Program

Type choices, press Enter.

Text 'description' . . . . . .   *SRCMBRTXT_____________________________________
_____

Compiler debugging dump
                                 1_____         1-32767, *
                                 32767_         1-32767
Intermediate text dump . . . . .  0_____        0-31









                                                                    Bottom
F3=Exit   F4=List   F5=Refresh   F11=Keywords   F12=Previous   F13=Prompter help
```

Figure 7. The Second CRTCBLPGM Prompt Screen

Note: Any CRTCBLPGM command default can be changed by using the CL command CHGCMDDFT. For more information, refer to the *CL Reference*.

If these parameter values are acceptable, press ENTER to process the command.

Press F3 to exit without processing the command.

Press F11 from either screen to display the corresponding parameter keywords and entry fields, as shown below.

```
CRTCBLPGM                  Create COBOL Program

Type choices, press Enter.

Program  . . . . . . . . . . . . PGM           *PGMID____
  Library  . . . . . . . . . . .                 QGPL_____
Source file  . . . . . . . . . . SRCFILE        QCBLSRC___
  Library  . . . . . . . . . . .                 *LIBL_____
Source member  . . . . . . . . . SRCMBR         *PGM______
Source listing options . . . . . OPTION         __________
                          + for more values     __________
Generation options . . . . . . . GENOPT         __________
                          + for more values     __________
Generation severity level  . . . GENLVL         29____
Print file . . . . . . . . . . . PRTFILE        QSYSPRT___
  Library  . . . . . . . . . . .                 *LIBL_____
FIPS flagging level  . . . . . . FIPS           *NO
Flagging severity  . . . . . . . FLAG           0_____
User profile . . . . . . . . . . USRPRF         *USER_
Public authority . . . . . . . . PUBAUT         *NORMAL
                                                                More...
F3=Exit   F4=List   F5=Refresh   F11=Choices   F12=Previous   F13=Prompter help
```

Figure 8. The First CRTCBLPGM Prompt Screen, Showing Keywords

```
CRTCBLPGM                  Create COBOL Program

Type choices, press Enter.

Text 'description' . . . . . . . TEXT          *SRCMBRTXT_____________________
_____________________

Compiler debugging dump          DUMP
                                               1_____
                                               32767_
Intermediate text dump . . . . . ITDUMP        0_____









                                                                Bottom
F3=Exit   F4=List   F5=Refresh   F11=Choices   F12=Previous   F13=Prompter help
```

Figure 9. The Second CRTCBLPGM Prompt Screen, Showing Keywords

Following are the descriptions of the keywords and the associated parameters and options. The defaults are underlined.

PGM

Specifies the qualified name by which the compiled COBOL program is known and the library in which the compiled program is to be located.

*PGMID
The name is taken from the PROGRAM-ID paragraph in the source program.

program-name
The first program in the batch job is to have this name, and all other programs are to use the name specified in the PROGRAM-ID paragraph in the source program.

QGPL
The name of the library in which the created program is stored if no library-name is specified.

library-name
Enter the name of any library in which the created program is to be stored.

SRCFILE

Specifies the name of the library and source file that contains the COBOL source program to be compiled.

QCBLSRC
The default source file, QCBLSRC, contains the COBOL source to be compiled.

source-file-name
Enter the name of the file that contains the COBOL source program to be compiled. This source file should have a record length of 92.

*LIBL
The system searches the library list to find the library in which the source file is located.

library-name
Enter the name of the library where the source file is stored.

SRCMBR

Specifies the name of the member of the source file that contains the COBOL source program to be compiled. This parameter can be specified only if the source file name in the SRCFILE parameter is a data base file.

*PGM
Use the name specified by the PGM parameter as the source file member name. If *PGMID is specified for the PGM parameter, the SRCMBR parameter is not used. For a data base source file, the first member is used.

source-file-member-name
Enter the name of the member that contains the COBOL source program.

OPTION

Specifies the options to use when the source program is compiled. Any or all of the following options can be specified in any order.

*SOURCE (or *SRC)
Produce a source listing, consisting of the COBOL source input and all compile-time error messages.

*NOSOURCE (or *NOSRC)
Do not produce a source listing.

*NOXREF
Do not produce a cross-reference listing for the source program.

*XREF
Produce a cross-reference listing for the source program.

*GEN
Create an object program.

*NOGEN
Do not create an object program.

*SEQUENCE
Check the reference numbers for sequence errors. Sequence errors do not occur if the *LINENUMBER option is specified.

*NOSEQUENCE
Do not check reference numbers for sequence errors.

*NOVBSUM
Do not print verb usage counts.

*VBSUM
Print verb usage counts.

*NONUMBER
The source file sequence numbers are used for reference numbers.

*NUMBER
The user-supplied sequence numbers (columns 1 through 6) are used for reference numbers.

*LINENUMBER
The compiler-generated sequence numbers are used for reference numbers. This option combines program source code and source code introduced by COPY statements into one consecutively numbered sequence. Use this option if you specify FIPS flagging.

*NOMAP
Do not list the Data Division map.

*MAP
List the Data Division map.

*NOOPTIONS
Do not list the options in effect for this compilation.

*OPTIONS
List the options in effect for this compilation.

*QUOTE
Use the quote (") as delimiter for nonnumeric literals and Boolean literals. This also specifies that the value of the figurative constant QUOTE has the EBCDIC value of a quote.

*APOST
Use the apostrophe (') as a delimiter for nonnumeric literals and Boolean literals. This also specifies that the value of the figurative constant QUOTE has the EBCDIC value of an apostrophe.

GENOPT
Specify the options to use when the object program is created. The listings produced by this parameter could be required if a problem occurs in COBOL. Any or all of the options can be specified in any order.

*NOLIST
Do not list IRP and associated hexadecimal code and any error messages.

*LIST
List IRP and associated hexadecimal code and any error messages.

*NOXREF
Do not produce a cross-reference listing of all objects defined in the IRP.

*XREF:
Produce a cross-reference listing of all objects defined in the IRP.

*NOPATCH
Do not reserve space in the compiled program for a program patch area.

PATCH
Reserve space in the compiled program for a program patch area. The program patch area can be used for debugging purposes.

*NODUMP
Do not list the program template.

*DUMP
List the program template.

*NOATR
Do not list the attributes for the IRP source.

*ATR
List the attributes for the IRP source.

*RANGE
Process run-time checks for subscript ranges, but not index ranges. Checks are also performed for substring operations in compiler-generated code.

*NORANGE
Do not perform run-time checks.

*UNREF
Unreferenced data items are included in the object program.

*NOUNREF
Unreferenced data items are not included in the object program. This reduces the number of ODT (Object Definition Table) entries used, allowing a larger program to be compiled. The unreferenced data items still appear in the cross-reference listings produced by specifying OPTION(*REF).

*NOOPTIMIZE
The compiler performs only standard optimizations for the program.

*OPTIMIZE
The compiler generates a program for possibly more efficient processing, which will possibly require less storage. However, specifying *OPTIMIZE can substantially increase the time required to compile a program.

GENLVL
Specifies whether an object program is to be generated depending on the severity-level value of errors encountered during compilation. If errors occur in a program with a severity level greater than the value specified in this parameter, an object program is not generated. For example, if you do not want an object program generated if you have messages with a severity level of 20 or greater, specify 19 in this parameter.

29
The default severity level if a value is not specified.

severity-level
Enter a two-digit number, 00 through 29.

PRTFILE
Specifies the name of the library and file to contain the compiler listing. The file should have a minimum record length of 132. If a file with a record length less than 132 is specified, information is lost.

QSYSPRT
If a file-name is not specified, the compiler listing is directed to the IBM-supplied file, QSYSPRT.

file-name
Enter the name of the file to which the compiler listing is directed.

*LIBL
The system searches the library list, *LIBL, to find the library in which the file is located.

library-name
Enter the name of the library in which the file is located.

FIPS

The source program is FIPS flagged for the following specified level. (Select the *LINENUMBER option to ensure that the reference numbers used in the FIPS flagging messages are unique.)

*NO The source program is not FIPS flagged.
*L FIPS flag for low level and higher.
*LI FIPS flag for low-intermediate level and higher.
*HI FIPS flag for high-intermediate level and higher.
*H FIPS flag for high level.

FLAG

Specifies the minimum severity level of messages to be printed.

00
All messages are to be printed.

severity-level
Enter a two-digit number that specifies the minimum severity level of messages that are to be printed. Messages that have severity levels of the specified value or higher are listed.

USRPRF

Specifies under which user profile the compiled COBOL program is to be run. The profile of either the program owner or the program user is used to run the program and control which objects can be used by the program (including what authority the program has for each object).

*USER
The program user's user profile is to be used when the program is run.

*OWNER:
The user profiles of both the program's owner and user are to be used when the program is run. The collective sets of object authority in both user profiles are to be used to find and access objects during the program's run. Any objects that are created during the program are owned by the program's user.

Note: Specify the USRPRF parameter to reflect the security requirements of your installation. The security facilities available on the AS/400 system are described in detail in the *Security Reference* and the *CL Reference*.

PUBAUT

Specifies what authority for the program and its description is being granted to the public. The authority can be altered for all or for specified users after program creation through the GRTOBJAUT (Grant Object Authority) or RVKOBJAUT (Revoke Object Authority) commands. (For further information on these commands, see the *CL Reference*.)

*NORMAL
Will be treated like *CHANGE. The public has only operational rights for the compiled program. Any user can run the program, but cannot change it or debug it.

*ALL:
The public has complete authorization for the program.

*NONE:
Will be treated like *EXCLUDE. The public cannot use the program.

Note: Specify the PUBAUT parameter to reflect the security requirements of your installation. The security facilities available on the AS/400 system are described in detail in the *Security Reference*. You may also reference the *CL Reference*.

TEXT
Lets you enter text that briefly describes the program and its function.

*SRCMBRTXT
Indicates that the text for the object being created is to be the same as the text for the data base file member containing the COBOL source program. If the source comes from a device or in-line file, specifying *SRCMBRTXT has the same effect as specifying *BLANK.

*BLANK
No text is specified.

'text'
Enter text that briefly describes the program and its function. It can be a maximum of 50 characters in length and must be enclosed in apostrophes. The apostrophes are not part of the 50-character string.

DUMP
An IBM COBOL debugging aid for IBM service personnel.

ITDUMP
An IBM COBOL debugging aid for IBM service personnel.

PROCESS Statement

The PROCESS statement is an optional part of the COBOL source program. It also lets you specify compile-time options. Options specified in the PROCESS statement override the corresponding options specified in the CRTCBLPGM command. The following table illustrates some of the equivalent PROCESS statement options and CRTCBLPGM CL command parameters and options.

PROCESS Statement Option	CRTCBLPGM	
	Parameter	Option
SOURCE	OPTION	*SOURCE
NOXREF	OPTION	*NOXREF
GEN	OPTION	*GEN
SEQUENCE	OPTION	*SEQUENCE
NOVBSUM	OPTION	*NOVBSUM
NONUMBER	OPTION	*NONUMBER
NOMAP	OPTION	*NOMAP
NOOPTIONS	OPTION	*NOOPTIONS
QUOTE	OPTION	*QUOTE
NOLIST	GENOPT	*NOLIST
GENLVL(nn)	GENLVL	nn
FIPS(xx)	FIPS	*xx
FLAG(nn)	FLAG	nn

The PROCESS statement must be placed before the first source statement in the COBOL program immediately preceding the IDENTIFICATION DIVISION header.

The format of the PROCESS statement is as follows:

Format

```
PROCESS option-1 [ option-2 ] . . . [ option-n ] [ . ]
```

The following rules apply to the PROCESS statement:

- The word PROCESS and all options must appear within positions 8 through 72. Position 7 must be left blank. The remaining positions can be used as in COBOL source statements, positions 1 through 6 for sequence numbers, positions 73 through 80 for identification purposes.
- Options must be separated by one or more blanks and/or commas.
- Options can appear in any order. If conflicting options are specified, for example, LIST and NOLIST, the last option encountered takes precedence.
- If the option keyword is correct and the suboption(s) is in error, the default suboption(s) is assumed.
- The PROCESS statement begins with the word PROCESS. Options can appear on more than one line; however, only the first line can contain the word PROCESS.

The allowable options for the PROCESS statement are listed below. Defaults are underlined. The descriptions for these options follow the Figure 8 on page 40 under the OPTION and GENOPT parameters.

SOURCE (or SRC)	NOMAP
NOSOURCE (or NOSRC)	MAP
NOXREF	NOOPTIONS
XREF	OPTIONS
GEN	QUOTE
NOGEN	APOST
SEQUENCE	NOLIST
NOSEQUENCE	LIST
NOVBSUM	GENLVL(nn)
VBSUM	
NONUMBER	FIPS(xx)
NUMBER	
LINENUMBER	FLAG(nn)

Figure 10 on page 50 illustrates how the PROCESS statement may be used.

Batch Compiles

The PROCESS statement is used to separate multiple programs and/or subprograms to be compiled with a single call of the compiler. In the batch compile environment, all compiler options specified on the CRTCBLPGM command statement, plus all default options, plus the options specified on the last PROCESS statement will be in effect for the compilation. All compiler output is directed to the destinations as specified by the CRTCBLPGM command statement.

All object programs are stored in the library specified on the PGM parameter. If program-name is specified for the PGM parameter, the first program in the batch job has that name, and all other programs use the name specified in the PROGRAM-ID paragraph in the source program.

Using COPY within the PROCESS Statement

The COBOL COPY statement can be used within the PROCESS statement to retrieve compiler options previously stored in a source library and include them in the PROCESS statement. COPY can be used to include options that override those specified as defaults by the compiler. Any PROCESS statement options can be retrieved with the COPY statement.

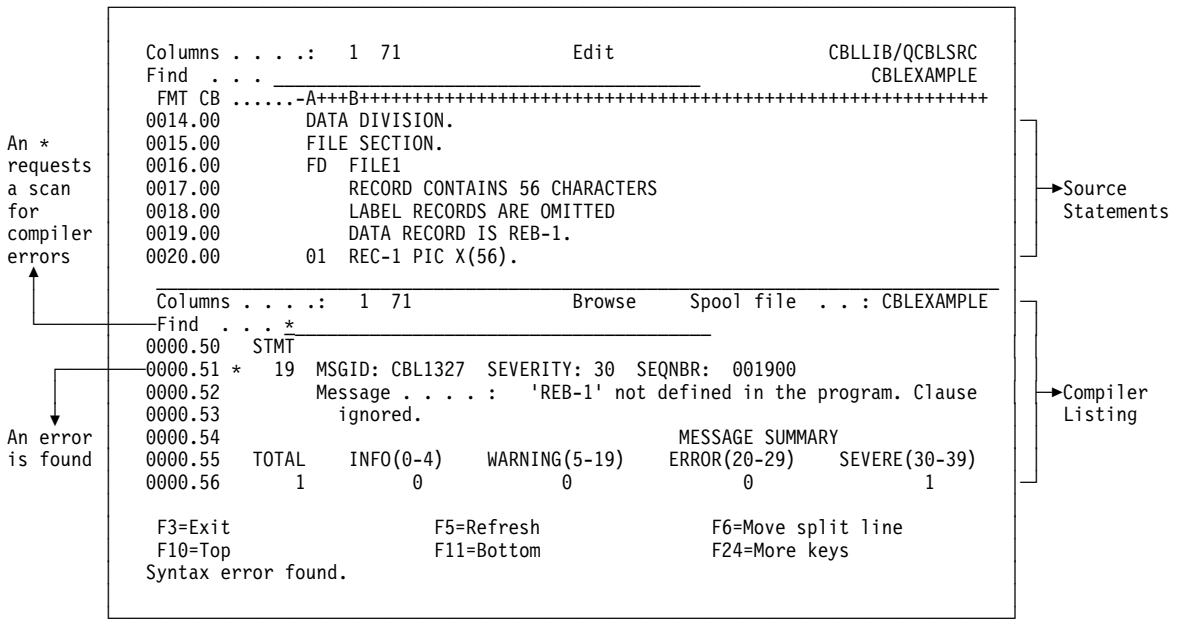
Compiler options can both precede and follow the COPY statement within the PROCESS statement. The last encountered occurrence of an option overrides all preceding occurrences of that option.

The following example shows the use of the COPY statement within the PROCESS statement. The COPY statement must be followed by a period. Notice also that in this example, NOMAP overrides the corresponding option in the library member:

```
000001  PROCESS XREF              MYPROG
000002  COPY DEFLTS.              MYPROG
        MAP, SOURCE, LIST         DEFLTS
000004  NOMAP, FLAG(20)           MYPROG
000010  IDENTIFICATION DIVISION.  MYPROG
```

Using SEU to Browse through a Compiler Listing

SEU lets you browse through a compiler listing that is on an output queue. The following shows SEU's split-edit display that lets you browse through the listing from a work station.



While browsing, you can:

- Scan for errors

- Correct source statements that have errors.

For complete information on browsing through a compiler listing, see the *SEU*.

Compiler Output

The result of compiling a program can include:

- A summary of command options.
- An options listing: A listing of options in effect for the compilation.
- A source listing: A listing of the statements contained in the source program.
- A verb usage listing: A listing of the COBOL verbs and the number of times each verb is used.
- A Data Division map: A glossary of compiler-generated information about the data. Also included is a mapping of user-supplied names to compiler-generated internal names.
- FIPS messages: A list of all FIPS messages for the requested FIPS level and above.
- A cross-reference list.
- Compiler messages (including diagnostic statistics).
- Compilation statistics.
- A listing of the generated program in symbolic form.
- An object program.

The presence or absence of some of these types of compiler output is determined by options specified on the PROCESS statement or through the CRTCBLPGM command. The level of diagnostic messages printed depends upon the FLAG option. Page ejection of the source program listing is obtained by placing the slash / character in the continuation area of a comment line.

Figure 11 on page 51 through Figure 16 on page 57 illustrate the compiler output produced for the example program. References to the figures are made throughout the following text. The letters in text correspond to the letters in the figures and reference an area of the output as it is being discussed.

Command Summary

This summary, which is output after compilation, lists all the compiler options specified in the CRTCBLPGM command statement and as modified by the PROCESS statement.

Listing of Compiler Options

The PROCESS statement, if specified, is printed immediately. Figure 10 on page 50 shows a list of all options in effect for the compilation of an example program. Compiler options are listed at the beginning of all compiler output when the OPTIONS option is specified.

```
5763CB1                              COBOL SOURCE LISTING
 STMT LINNBR -A 1 B.. ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 .IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 PROCESS  SOURCE  XREF  GEN  NOSEQUENCE VBSUM   LINENUMBER MAP
    2  000001          OPTIONS APOST LIST GENLVL(19) FIPS(H) FLAG(20)
                    COBOL COMPILER OPTIONS IN EFFECT
                        OPTIONS
                        SOURCE
                        XREF
                        MAP
                        VBSUM
                        LINENUMBER
                        NOSEQUENCE
                        GEN
                        GENLVL(19)
                        FLAG(20)
                        FIPS(H)
                        APOST
                    COBOL GENERATION OPTIONS IN EFFECT
                        LIST
                        UNREF
                        RANGE
                        NOATR
                        NOXREF
                        NODUMP
                        NOPATCH
                        NOOPTIMIZE
```

Figure 10. Listing of Compiler Options

Source Listing

Figure 11 on page 51 illustrates a source listing. The statements in the source program are listed exactly as submitted. The source is not listed if the NOSOURCE option is specified.

All compiler output pages after the page where the PROGRAM-ID paragraph is listed have the program-id name listed in the heading prior to the date field. Figure 11 on page 51 displays the following fields:

A *Compiler-generated statement number:* The numbers appear to the left of the source program listing. These numbers are referenced in all compiler output listings except for FIPS messages listings. A statement number can span several lines, and a line can contain more than one statement.

B *Reference number:* The numbers appear to the left of the source statements. The numbers that appear in this field and the column heading (shown as SEQNBR in this listing) are determined by an option specified in the CRTCBLPGM command or in the PROCESS statement, as shown in the following table.

Option	Heading	Origin
NONUMBER	SEQNBR	Source file sequence numbers.
NUMBER	NUMBER	Standard COBOL sequence numbers.
LINENUMBER	LINNBR	Compiler generated sequence numbers.

C *Sequence error indicator column:* An S in this column indicates that the line is out of sequence. Sequence checking is performed on the reference number field only if the SEQUENCE option is specified.

D *Copyname:* The copyname, as specified in the COBOL COPY statement, is listed here for all records included in the source program by that COPY statement. If the DDS-ALL-FORMATS phrase is used, the name '<--ALL-FMTS' appears under copyname.

E *Change/date field:* The date the line was last modified is listed here.

```
5763CB1                              COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  A     B                                                                              C     D        E
    3  000300 IDENTIFICATION DIVISION.                                                            05/05/94
    4  000400 PROGRAM-ID.     EXMPLE-PROGRAM.                                                     05/05/94
    5  000500   AUTHOR.       PROGRAMMER NAME.                                                    05/05/94
    6  000600   INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.                                   05/05/94
    7  000700   DATE-WRITTEN. 09/07/88.                                                           05/05/94
    8  000800   DATE-COMPILED.  09/07/88 12:54:11   .                                           05/05/94
    9  000900 ENVIRONMENT DIVISION.                                                              05/05/94
   10  001000 CONFIGURATION SECTION.                                                             05/05/94
   11  001100 SOURCE-COMPUTER. IBM-S38.                                                          05/05/94
   12  001200 OBJECT-COMPUTER. IBM-S38.                                                          05/05/94
   13  001300 INPUT-OUTPUT SECTION.                                                              05/05/94
   14  001400 FILE-CONTROL.                                                                      05/05/94
   15  001500     SELECT FILE-1 ASSIGN TO DISK-EXMPLE.                                           05/05/94
   16  001600 DATA DIVISION.                                                                     05/05/94
   17  001700 FILE SECTION.                                                                      05/05/94
   18  001800 FD  FILE-1                                                                         05/05/94
   19  001900     LABEL RECORDS ARE STANDARD                                                     05/05/94
   20  002000     RECORD CONTAINS 20 CHARACTERS                                                  05/05/94
   21  002100     DATA RECORD IS RECORD-1.                                                       05/05/94
   22  002200 01  RECORD-1.                                                                    05/05/94
   23  002300   02 FIELD-A      PIC X(20).                                                      05/05/94
   24  002400 WORKING-STORAGE SECTION.
   25  002500 01  FILLER.
   26  002600   05 KOUNT        PIC S9(2) COMP-3.
   27  002700   05 ALPHABET     PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
   28  002800   05 ALPHA REDEFINES ALPHABET
   29  002900                   PIC X(1)  OCCURS 26 TIMES.
   30  003000   05 NUMBR        PIC S9(2) COMP-3.
   31  003100   05 DEPENDENTS   PIC X(26) VALUE "01234012340123401234012340".
   32  003200   05 DEPEND REDEFINES DEPENDENTS
   33  003300                   PIC X(1)  OCCURS 26 TIMES.
   34  003400 COPY WRKRCD.
   35 +000100 01  WORK-RECORD.                                                                  WRKRCD
   36 +000200   05 NAME-FIELD   PIC X(1).                                                       WRKRCD
   37 +000300   05 FILLER       PIC X(1)  VALUE SPACE.                                         WRKRCD
   38 +000400   05 RECORD-NO    PIC S9(3).                                                     WRKRCD
   39 +000500   05 FILLER       PIC X(1)  VALUE SPACE.                                         WRKRCD
   40 +000600   05 LOCATION     PIC A(3)  VALUE "NYC".                                         WRKRCD
   41 +000700   05 FILLER       PIC X(1)  VALUE SPACE.                                         WRKRCD
   42 +000800   05 NO-OF-DEPENDENTS                                                           WRKRCD
   43 +000900                   PIC X(2).                                                      WRKRCD
   44 +001000   05 FILLER       PIC X(7)  VALUE SPACES.                                        WRKRCD
      003500*******************************************************
      003600* THE FOLLOWING PARAGRAPH OPENS THE OUTPUT FILE TO  *
      003700* BE CREATED AND INITIALIZES COUNTERS               *
      003800*******************************************************
   45  003900 PROCEDURE DIVISION.
      004000 STEP-1.
   46  004100     OPEN OUTPUT FILE-1.
   47  004200     MOVE ZERO TO KOUNT, NUMBR.
      004300*******************************************************
      004400* THE FOLLOWING 3 PARAGRAPHS CREATE INTERNALLY THE  *
      004500* RECORDS TO BE CONTAINED IN THE FILE, WRITE THEM   *
      004600* ON THE DISK, AND DISPLAY THEM                     *
      004700*******************************************************
```

Figure 11 (Part 1 of 2). Source Listing

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      004800 STEP-2.
   48 004900     ADD 1 TO KOUNT, NUMBR.
   49 005000     MOVE ALPHA  (KOUNT) TO NAME-FIELD.
   50 005100     MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
   51 005200     MOVE NUMBR          TO RECORD-NO.
      005300 STEP-3.
   52 005400     DISPLAY WORK-RECORD.
   53 005500     WRITE RECORD-1 FROM WORK-RECORD.
      005600 STEP-4.
   54 005700     PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
      005800*****************************************************
      005900* THE FOLLOWING PARAGRAPH CLOSES FILE OPENED FOR    *
      006000* OUTPUT AND RE-OPENS IT FOR INPUT                  *
      006100*****************************************************
      006200 STEP-5.
   55 006300     CLOSE FILE-1.
   56 006400     OPEN INPUT FILE-1.
      006500*****************************************************
      006600* THE FOLLOWING PARAGRAPHS READS BACK THE FILE AND  *
      006700* SINGLES OUT EMPLOYEES WITH NO DEPENDENTS          *
      006800*****************************************************
      006900 STEP-6.
   57 007000     READ FILE-1 RECORD INTO WORK-RECORD
   58 007100       AT END GO TO STEP-8.
      007200 STEP-7.
   59 007300     IF NO-OF-DEPENDENTS IS EQUAL TO "0"
   60 007400       MOVE "Z" TO NO-OF-DEPENDENTS.
   61 007500     GO TO STEP-6.
      007600 STEP-8.
   62 007700     CLOSE FILE-1.
   63 007800     STOP RUN.
                        * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 11 (Part 2 of 2). Source Listing

Verb Usage by Count Listing

Figure 12 shows the alphabetic list that is produced of all verbs used in the source program. A count of how many times each verb was used is also included. This listing is produced when the VBSUM option is specified.

```
5763CB1                              COBOL VERB USAGE BY COUNT

VERB                       COUNT

ADD                            1
CLOSE                          2
DISPLAY                        1
GO                             2
IF                             1
MOVE                           5
OPEN                           2
PERFORM                        1
READ                           1
STOP                           1
WRITE                          1
                     * * * * *  E N D  O F  V E R B  U S A G E  * * * * *
```

Figure 12. Verb Usage by Count Listing

Data Division Map

The Data Division map, Figure 13 on page 54, is listed when the MAP option is specified. The Data Division map contains information about names in the COBOL source program. The number of bytes required for the File Section and Working-Storage Section is given at the end of the Data Division map.

Figure 13 on page 54 displays the following fields:

F *Statement number:* The compiler-generated statement number where the data item was defined is listed for each data item in the Data Division map.

G *Level of data item:* The level-number of the data item, as specified in the source program, is listed here. Index-names are identified by an *IX* in the level-number and a blank type field.

H *Source name:* The data-name as specified in the source program is listed here.

I *Section:* The section in which the item was defined is shown here through the use of the following codes:

- `FS` File Section
- `WS` Working-Storage Section
- `LS` Linkage Section
- `SM` Sort/Merge Section
- `SR` Special Register

J *Displacement:* The offset, in bytes, of the item from the level-01 group item is given here.

K *Length:* The decimal length of the item is listed here.

L *Type:* The data class type for the item is shown here through the use of the following codes:

GROUP	Group Item
`A`	Alphabetic
`AE`	Alphabetic edited
`AN`	Alphanumeric
`ANE`	Alphanumeric edited
`INDEX`	Index data item (USAGE INDEX)
`BOOLN`	Boolean
`ZONED`	Zoned decimal
`PACKED`	Packed decimal (COMP or COMP-3)
`BINARY`	Binary (COMP-4)
`NE`	Numeric edited

M *Internal name:* The compiler-generated internal names are listed here and are assigned as follows:

File-names The internal name uses the form .Fnn, where .F indicates a file-name, and nn is a unique two-digit number.

Data-names The internal name uses the form .Dxxxxxx, where .D indicates a data-name or index-name, and xxxxxx is a unique six-digit hex value. These names appear on the `PRM` listing if generated.

N *Attributes:* The attributes of the item are listed here as follows:

- For files, the following information can be given:
  - Device type
  - `ORGANIZATION`
  - `ACCESS MODE`
  - `BLOCK CONTAINS` information
  - `RECORD CONTAINS` information

  - LABEL information
  - RERUN specified is indicated
  - SAME AREA specified is indicated
  - CODE-SET specified is indicated
  - SAME RECORD AREA specified is indicated
  - LINAGE specified is indicated.

- For data items, the attributes indicate whether the following information was specified for the item:

  - REDEFINES
  - VALUE
  - JUSTIFIED
  - SYNCHRONIZED
  - BLANK WHEN ZERO
  - SIGN IS LEADING
  - SIGN IS LEADING SEPARATE
  - SIGN IS SEPARATE
  - INDICATORS.

- For table items, the dimensions for the item are listed here in the form dim ( ). For each dimension, a maximum OCCURS value is given. When a dimension is variable, it is listed as such, giving the lowest and highest OCCURS values.

```
5763CB1                          COBOL DATA DIVISION MAP

STMT  LVL  SOURCE NAME                SECTION  DISP   LEN   TYPE    I-NAME    ATTRIBUTES
 F     G       H                        I       J      K     L        M          N

   18  FD  FILE-1                        FS                          .F01      DEVICE DISK, ORGANIZATION SEQUENTIAL,
                                                                              ACCESS SEQUENTIAL, BLOCK CONTAINS 20
                                                                              CHARACTERS, RECORD CONTAINS 20
                                                                              CHARACTERS, LABEL RECORDS STANDARD
   22  01  RECORD-1                     FS  000000    20  GROUP   .D005454
   23  02  FIELD-A                      FS  000000    20  AN      .D0054A8
   25  01  FILLER                       WS  000000    56  GROUP   .D0054FC
   26  02  KOUNT                        WS  000000     2  PACKED  .D00554E
   27  02  ALPHABET                     WS  000002    26  AN      .D0055B2  VALUE
   28  02  ALPHA                        WS  000002     1  AN      .D00562E  REDEFINES .D0055B2, DIMENSION(26)
   30  02  NUMBR                        WS  000028     2  PACKED  .D00568E
   31  02  DEPENDENTS                   WS  000030    26  AN      .D0056F2  VALUE
   32  02  DEPEND                       WS  000030     1  AN      .D005770  REDEFINES .D0056F2, DIMENSION(26)
   35  01  WORK-RECORD                  WS  000000    19  GROUP   .D0057D0
   36  02  NAME-FIELD                   WS  000000     1  AN      .D005828
   37  02  FILLER                       WS  000001     1  AN      .D00587E  VALUE
   38  02  RECORD-NO                    WS  000002     3  ZONED   .D0058D8
   39  02  FILLER                       WS  000005     1  AN      .D005940  VALUE
   40  02  LOCATION                     WS  000006     3  A       .D00599A  VALUE
   41  02  FILLER                       WS  000009     1  AN      .D005A00  VALUE
   42  02  NO-OF-DEPENDENTS             WS  000010     2  AN      .D005A5A
   44  02  FILLER                       WS  000012     7  AN      .D005AB6  VALUE
FILE SECTION uses 20 bytes of storage
WORKING-STORAGE SECTION uses 75 bytes of storage
                 * * * * *  E N D  O F  D A T A  D I V I S I O N  M A P  * * * * *
```

Figure 13. Data Division Map

FIPS Messages

The FIPS messages, Figure 14 on page 55, are listed when the FIPS option is specified. Only messages for the requested FIPS level and above are listed. Figure 14 displays the following fields:

```
5763CB1                                COBOL FIPS MESSAGES

 FIPS-ID    DESCRIPTION AND SEQUENCE NUMBERS FLAGGED
  O                        P

CBL8200     Following items only valid at FIPS low-intermediate level or higher. Q
CBL8201     COPY statement.
                 003400
CBL8300     Following items valid only at FIPS high-intermediate level or higher.
CBL8302     Plural form of figurative constants used.
                 001000
CBL8303     DATE-COMPILED paragraph.
                 000800
CBL8331     Multiple receivers in ADD, SUBTRACT, MULTIPLY, or DIVIDE statement.
                 004900
CBL8350     UNTIL phrase of PERFORM statement.
                 005700
CBL8360     Comparison of nonnumeric operands of unequal size.
                 007300
CBL8374     Comma or semicolon as punctuation.
                 004200 004900
CBL8500     Following items are IBM-defined or are IBM extensions. Q
CBL8504     Assignment-name in ASSIGN clause.
                 001500
CBL8506     FILLER used as group item.
                 002500
CBL8518     USAGE IS COMPUTATIONAL-3.
                 002600 003000
CBL8561     COPY statement with default library assumed.
                 003400
   13 FIPS violations flagged. R
                 * * * * *  E N D  O F  F I P S  M E S S A G E S  * * * * *
```

Figure 14. FIPS Messages

O FIPS-ID: This field lists the FIPS message number.

P *Description and reference numbers flagged:* This field lists a description of the condition flagged, followed by a list of the reference numbers from the source program where this condition is found. The type of reference numbers used, and their names in the heading (shown as SEQUENCE NUMBERS in this listing) are determined by an option specified in the CRTCBLPGM command or in the PROCESS statement, as shown in the following table.

Option	Heading
NONUMBER	DESCRIPTION AND SEQUENCE NUMBERS FLAGGED
NUMBER	DESCRIPTION AND USER-SUPPLIED NUMBERS FLAGGED
LINENUMBER	DESCRIPTION AND LINENUMBERS FLAGGED

Q *Items grouped by level:* These headings subdivide the FIPS messages by level.

R FIPS *violations flagged:* The total number of FIPS violations flagged is included at the end of the FIPS listing.

Cross-Reference List

The cross-reference list, Figure 15, is produced when the XREF option is specified. It provides a list of all data references and procedure-name references, by statement number, within the source program. Figure 15 displays the following fields:

```
5763CB1                          COBOL CROSS REFERENCE LISTING

NAMES (* = Procedure-name)       DEFINED  REFERENCES (* = Changed)
  S                                T        U

 ALPHA                             28    49
 ALPHABET                          27    28
 DEPEND                            32    50
 DEPENDENTS                        31    32
*DUMMY-SECTION                     46
 FIELD-A                           23
 FILE-1                            18    15  46  55  56  57  62
 KOUNT                             26    47* 48* 49  50  54
 LOCATION                          40
 NAME-FIELD                        36    49*
 NO-OF-DEPENDENTS                  42    50* 59  60*
 NUMBR                             30    47* 48* 51
 RECORD-NO                         38    51*
 RECORD-1                          22    21  53*
*STEP-1                            46
*STEP-2                            48    54
*STEP-3                            52    54
*STEP-4                            54
*STEP-5                            55
*STEP-6                            57    61
*STEP-7                            59
*STEP-8                            62    58
 WORK-RECORD                       35    52  53  57*
                  * * * * *  E N D  O F  C R O S S  R E F E R E N C E  * * * * *
```

Figure 15. Cross-Reference List

S *Names field:* The data-name or procedure-name referenced is listed here. All procedure-names are flagged with an * before the name. The names are listed alphabetically.

T *Defined field:* The statement number where the name was defined within the source program is listed here.

U *References field:* All statement numbers are listed in the same sequence as the name is referenced in the source program. An * following a statement number indicates that the item was modified in that statement.

Messages

Figure 16 shows the messages that are generated during program compilation. The fields displayed are:

```
 5763CB1                                  COBOL MESSAGES

  STMT
   V
        X                                W
*       MSGID: CBL0904  SEVERITY: 00  SEQNBR:
        Message . . . . :   Unexpected source member type.  Y
*    4  MSGID: CBL0047  SEVERITY: 20  SEQNBR:  000130
        Message . . . . :   Invalid program-name 'EXMPLE-PROGRAM'.  Y
          Accepted as 'EXMPLE0PRO'.
*   18  MSGID: CBL0650  SEVERITY: 00  SEQNBR:  000270
        Message . . . . :   Blocking/Deblocking for file 'FILE-1' will  Y
          be performed by compiler-generated code.
    Z                                    MESSAGE SUMMARY
  TOTAL    INFO(0-4)    WARNING(5-19)    ERROR(20-29)    SEVERE(30-39)    TERMINAL(40-99)
      3          2              0               1               0                0
                  * * * * *  E N D  O F  C O B O L  M E S S A G E S  * * * * *
```

Figure 16. Diagnostic Messages

V *Statement number:* This field lists the compiler-generated statement number associated with the statement in the source program for which the message was issued.[3]

W *Reference number:* The reference number is issued here.[3] The numbers that appear in this field are determined by an option specified in the CRTCBLPGM command or in the PROCESS statement, as shown in the following table.

Option	Heading	Origin
NONUMBER	SEQNBR	Source file sequence numbers
NUMBER	NUMBER	Standard COBOL sequence numbers
LINENUMBER	LINNBR	Compiler generated sequence numbers

When a message is issued for a record from a copy file, the number is preceded by a + or a -.

X *MSGID and Severity Level:* These fields contain the message number and its associated severity level. Severity levels are defined as follows:

- 00 Informational
- 10 Warning
- 20 Conditional
- 30 Severe
- 40 Unrecoverable

Y *Message:* The message identifies the condition and indicates the action taken by the compiler.

[3] The statement number and the reference number do not appear on certain messages that reference missing items. For example, if the member TYPE (CBL38) is not specified, message CBL0904 appears on the listing with no statement or reference number listed.

Z *Message statistics:* This field lists the total number of messages and the number of messages by severity level.

The totals listed are the number of messages generated for each severity by the compiler and will not always be the number listed. For example, if `FLAG(10)` is specified, no messages of severity less than 10 are listed. However, the counts will indicate the number that would have been printed if they had not been suppressed.

Chapter 4. Running and Debugging Your Program

There are many ways to run a COBOL program depending on how the program is written and who is using the program. See the *CL Programmer's Guide* for the various ways. The three most common ways are through:

- A control language CALL statement or the COBOL CALL statement
- An application-oriented menu
- A user-created command.

A control language CALL statement can be part of a batch job, entered interactively by a work station user, or included in a CL program. An example of a control language CALL statement is CALL PAYROLL. Payroll is the name of a COBOL program that is called and then run.

A COBOL program can call another program with the COBOL CALL statement (see Chapter 11, “Using the Additional COBOL Functions”). Another way to run a COBOL program is through an application-oriented menu. The work station user can request an application-oriented menu and then select an option that will call the appropriate program. The following is an example of an application-oriented menu:

```
                          PAYROLL DEPARTMENT MENU

1.  Inquire into employee master

2.  Change employee master

3.  Add new employee

4.  Return


Option:__
```

This menu is normally displayed by a control language program in which each option calls a separate COBOL program. When a COBOL program ends, the system returns control to whatever called the program. This could be a work station user, a CL program (such as the menu handling program), or another COBOL program.

You can also create a command to run a COBOL program by using a command definition. Refer to the *CL Programmer's Guide* for information on using the command definition. For example, you can create a command, PAY, which calls a program, PAYROLL. A user-created command can be entered into a batch job, or it can be entered interactively by a work station user.

If the Program Ends Abnormally

If a COBOL program ends abnormally while being run, the escape message CBE9001 is issued. A CL program can monitor for this exception by using the MONMSG (Monitor Message) command.

If a program ends for some reason other than by encountering a STOP statement or falling through to the end of the program, the return code is set to 2. See the RTVJOBA and DSPJOB commands in the *CL Reference* for more information about return codes.

Requesting Data from Job Stream

When the user is running a batch job that uses the ACCEPT statement, the input data is taken from the job stream. The data must be placed immediately following the CL command CALL for the COBOL program. It is the user's responsibility to request (through ACCEPTs) the same amount of data as is available.

Note: If more data is requested than is available, the CL statement following the data is treated as input data. If more data is available than is requested, each extra line of data is treated as a CL statement. In either of the above cases, undesirable results can occur.

Debugging Your Program

COBOL and OS/400 provide functions that you can use to test and debug the programs you develop:

OS/400 Functions	COBOL Functions
Test library	Debugging features
Breakpoints	Formatted dump
Traces	

The OS/400 functions let you test programs while optionally protecting your production files, and let you observe and debug operations as a program runs. No special source code is required for using the OS/400 functions.

The COBOL functions can be used independently of the OS/400 functions or in combinations with them to:

- Debug a program
- Produce a formatted dump of the contents of fields, data structures, arrays, and tables.

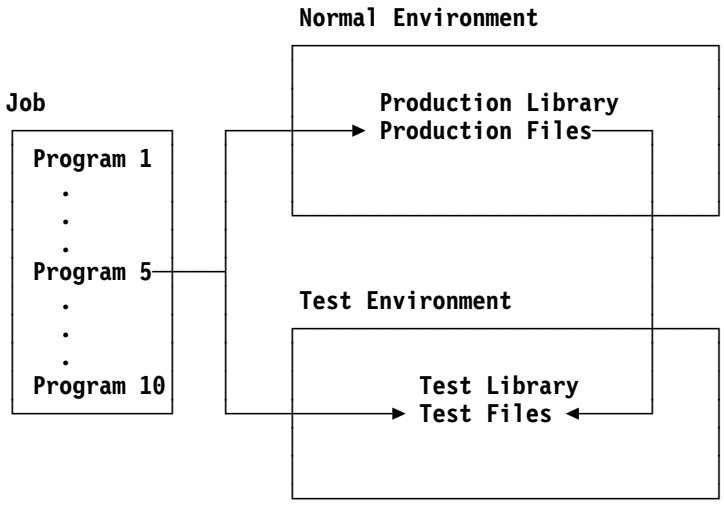
Source code is required for using the COBOL Debugging features and formatted dump capability. A formatted dump can also be obtained by a user's response to a run-time message.

OPEN-FEEDBACK and I-O-FEEDBACK contents can provide additional debugging information. The method for obtaining this information is described later in this chapter.

Using a Test Library

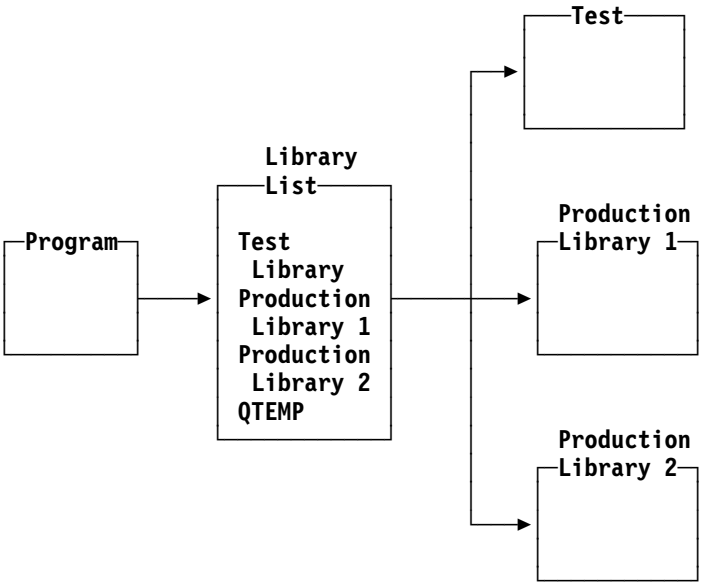
The basic concept of testing and debugging is that of a testing environment. Programs running in a normal operating environment can read, update, and write records that are in either test or production libraries. Programs running in a test environment can read, update, and write records in either test or production libraries. However, to prevent data base files in production libraries from being modified unintentionally, you can specify the UPDPROD(*NO) parameter on the Enter Debug (ENTDBG) command or Change Debug (CHGDBG) command (see the *CL Reference).*

On the AS/400 system, you can copy production files into the test library or you can create special files for testing in this library. A test copy of a file and its production copy can have the same name if the files are in different libraries. You can use the same file name in the program for either testing or normal processing.

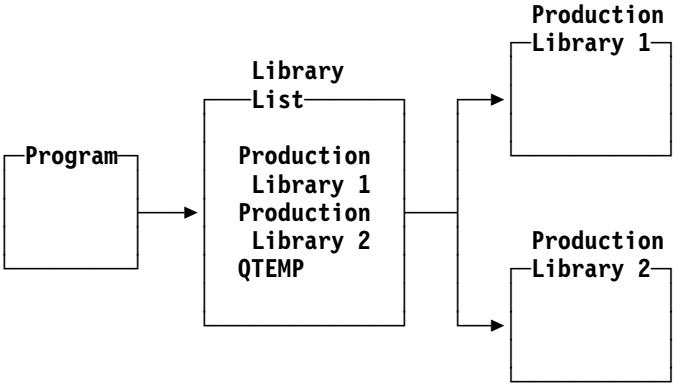


For testing, you must place the test library name ahead of the production library name in the library list for the job that contains the program to be tested. For normal processing, the test library should not be named in the library list for that job.

Testing a Program

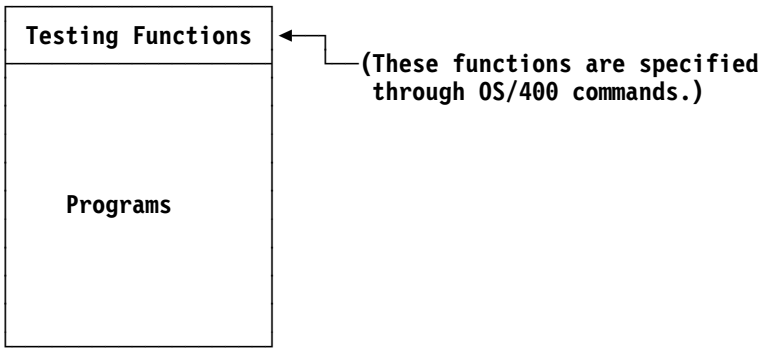


Running a Program Normally



No special statements for testing are contained within the program being tested. The same program being tested can be run normally without modifications. All testing functions are specified within the job that contains the program and not within the program.

Using the Same Program in Several Jobs



Testing functions apply only to the job in which they are specified. A program can be used concurrently in two jobs: one job that is in a test environment and another job that is in a normal processing environment.

Testing functions of OS/400 let you interact with a program as it runs to observe the operations being done. These functions include using breakpoints and traces.

Using Breakpoints

A breakpoint is a statement number or a label in your program where you want the program to stop. If you use a statement number, it should be a statement number that appears on the compiler listing of the COBOL source program. If you use a label as a breakpoint rather than a statement number, the label can be:

- Associated with a function performed by your COBOL program (for example, .OPEN indicates the open file function).
- An internal COBOL compiler generated label (for example, .L000001 indicates the first internally generated label).

Note: To determine the internally generated labels for your program, use the GENOPT parameter on the CRTCBLPGM command to get an IRP listing of the program.

When a breakpoint statement is about to be processed for an interactive job, the system displays the breakpoint at which the program has stopped and, if requested, the values of program variables. After you get this information (in a display), you can go to a Command Entry screen and then enter CL commands to request other functions (such as displaying or changing a variable, adding a breakpoint, or adding a trace). You can then cancel the program (CNLRQS) or resume processing from the breakpoint (RSMBKP).

For a batch job, a breakpoint program can be called when a breakpoint is reached. The breakpoint information is passed to the breakpoint program.

Example of Using Breakpoints

Figure 17 on page 64 shows an example COBOL program, TESTPRT. The following CL commands add breakpoints at statements 43 and 52. The value of variable KOUNT is displayed when the breakpoint at statement 52 is reached.

CL Commands:

```
ENTDBG      TESTPRT
ADDBKP      STMT(43)
ADDBKP      STMT(52)
            PGMVAR(KOUNT)
```

All CL commands are explained in the *CL Reference.*

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.     TESTPRT.
    3  000300   AUTHOR.       PROGRAMMER NAME.
    4  000400   INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
    5  000500   DATE-WRITTEN. 08/30/88.
    6  000600   DATE-COMPILED.  08/31/88 15:15:54   .
    7  000700 ENVIRONMENT DIVISION.
    8  000800 CONFIGURATION SECTION.
    9  000900 SOURCE-COMPUTER. IBM-S38.
   10  001000 OBJECT-COMPUTER. IBM-S38.
   11  001100 INPUT-OUTPUT SECTION.
   12  001200 FILE-CONTROL.
   13  001300     SELECT FILE-1 ASSIGN TO DISK-EXMPLE.
   14  001400 DATA DIVISION.
   15  001500 FILE SECTION.
   16  001600 FD  FILE-1
   17  001700     LABEL RECORDS ARE STANDARD
   18  001800     RECORD CONTAINS 20 CHARACTERS
   19  001900     DATA RECORD IS RECORD-1.
   20  002000 01  RECORD-1.
   21  002100   02 FIELD-A      PIC X(20).
   22  002200 WORKING-STORAGE SECTION.
   23  002300 01  FILLER.
   24  002400   05 KOUNT        PIC S9(2) COMP-3.
   25  002500   05 ALPHABET     PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
   26  002600   05 ALPHA REDEFINES ALPHABET
   27  002700                   PIC X(1)  OCCURS 26 TIMES.
   28  002800   05 NUMBR        PIC S9(2) COMP-3.
   29  002900   05 DEPENDENTS   PIC X(26) VALUE "01234012340123401234012340".
   30  003000   05 DEPEND REDEFINES DEPENDENTS
   31  003100                   PIC X(1)  OCCURS 26 TIMES.
   32  003200 01  WORK-RECORD.
   33  003300   05 NAME-FIELD   PIC X(1).
   34  003400   05 FILLER       PIC X(1)  VALUE SPACE.
   35  003500   05 RECORD-NO    PIC S9(3).
   36  003600   05 FILLER       PIC X(1)  VALUE SPACE.
   37  003700   05 LOCATION     PIC A(3)  VALUE "NYC".
   38  003800   05 FILLER       PIC X(1)  VALUE SPACE.
   39  003900   05 NO-OF-DEPENDENTS
   40  004000                   PIC X(2).
   41  004100   05 FILLER       PIC X(7)  VALUE SPACES.
       004200*******************************************************
       004300* THE FOLLOWING PARAGRAPH OPENS THE OUTPUT FILE TO  *
       004400* BE CREATED AND INITIALIZES COUNTERS               *
       004500*******************************************************
   42  004600 PROCEDURE DIVISION.
       004700 STEP-1.
--->43  004800     OPEN OUTPUT FILE-1.
   44  004900     MOVE ZERO TO KOUNT, NUMBR.
       005000*******************************************************
       005100* THE FOLLOWING 3 PARAGRAPHS CREATE INTERNALLY THE  *
       005200* RECORDS TO BE CONTAINED IN THE FILE, WRITES THEM  *
       005300* ON THE DISK, AND DISPLAYS THEM                    *
       005400*******************************************************
       005500 STEP-2.
```

Figure 17 (Part 1 of 2). Example of Using Breakpoints

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   45  005600     ADD 1 TO KOUNT, NUMBR.
   46  005700     MOVE ALPHA  (KOUNT) TO NAME-FIELD.
   47  005800     MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
   48  005900     MOVE NUMBR          TO RECORD-NO.
       006000 STEP-3.
   49  006100     DISPLAY WORK-RECORD.
   50  006200     WRITE RECORD-1 FROM WORK-RECORD.
       006300 STEP-4.
   51  006400     PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
       006500**************************************************
       006600* THE FOLLOWING PARAGRAPH CLOSES FILE OPENED FOR    *
       006700* OUTPUT AND RE-OPENS IT FOR INPUT                  *
       006800**************************************************
       006900 STEP-5.
--->52  007000     CLOSE FILE-1.
   53  007100     OPEN INPUT FILE-1.
       007200**************************************************
       007300* THE FOLLOWING PARAGRAPHS READS BACK THE FILE AND  *
       007400* SINGLES OUT EMPLOYEES WITH NO DEPENDENTS          *
       007500**************************************************
       007600 STEP-6.
   54  007700     READ FILE-1 RECORD INTO WORK-RECORD
   55  007800       AT END GO TO STEP-8.
       007900 STEP-7.
   56  008000     IF NO-OF-DEPENDENTS IS EQUAL TO "0"
   57  008100       MOVE "Z" TO NO-OF-DEPENDENTS.
   58  008200     GO TO STEP-6.
       008300 STEP-8.
   59  008400     CLOSE FILE-1.
   60  008500     STOP RUN.
                         * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 17 (Part 2 of 2). Example of Using Breakpoints

The first breakpoint is used to indicate where you are in the program. The following is displayed as a result of reaching the first breakpoint.

```
                              Display Breakpoint

 Statement/Instruction . . . . . . . . . :   43 /0018
 Program . . . . . . . . . . . . . . . . :   TESTPRT
 Recursion level . . . . . . . . . . . . :   1









 Press Enter to continue.

 F3=Exit program   F10=Command entry
```

The following is displayed as a result of reaching the second breakpoint.

```
                            Display Breakpoint

 Statement/Instruction . . . . . . . . . : 52 /0049
 Program . . . . . . . . . . . . . . . . : TESTPRT
 Recursion level . . . . . . . . . . . . : 1
 Start position  . . . . . . . . . . . . : 1
 Format  . . . . . . . . . . . . . . . . : *CHAR
 Length  . . . . . . . . . . . . . . . . : *DCL

 Variable  . . . . . . . . . . . . . . . : 05  KOUNT
   Type  . . . . . . . . . . . . . . . . :   PACKED
   Length  . . . . . . . . . . . . . . . :   2 0
  ' 26'









Press Enter to continue.

F3=Exit program   F10=Command entry
```

When specifying a variable for the PGMVAR parameter, every name must begin with an alphanumeric character (A through Z, $, #, or @) and can be followed by the characters (A through Z, 0 though 9, $, #, @, or _).

The following example shows how to display a COBOL variable, RECORD-NO, in the example program. Because the hyphen is treated by OS/400 as a special character, RECORD-NO must be enclosed in apostrophes.

```
ENTDBG     TESTPRT
ADDBKP     STMT(52)
           PGMVAR('RECORD-NO')
```

To display the value of a table element, the appropriate occurrence numbers (subscripts) must be included with the variable name. Up to three dimensions of subscripting are allowed, and the subscripts must be separated by commas.

Do not use an index-name or index data-item as a subscript. When an index is entered as a subscript, the operating system uses the internal value of the index as the subscript, and undesirable results can occur.

The following example shows how to specify the COBOL variable TABLE1 with three dimensions.

```
PGMVAR ('TABLE1 (SUB1, SUB2, SUB3)').
```

One or more blanks are allowed after each comma separating subscripts, but the total length of the variable plus subscripts, parentheses, commas, and blanks specified with the PGMVAR keyword cannot exceed 132 characters. For more information on how to code variables in CL commands, see the *CL Reference*.

Variable names can be qualified in the PGMVAR parameter, for example, NAME-FIELD OF WORK-RECORD in Figure 17 on page 64.

Another technique can be used to display variables that are not elements of a multi-dimensional table. For example, to display the field KOUNT you can use COBOL Data Division map to find its COBOL internal name (I-NAME).

```
5763CB1                                COBOL DATA DIVISION MAP
 STMT  LVL  SOURCE NAME                    SECTION  DISP  LEN   TYPE   I-NAME    ATTRIBUTES
    16  FD  FILE-1                             FS                      .F01      DEVICE DISK, ORGANIZATION SEQUENTIAL,
                                                                                 ACCESS SEQUENTIAL, BLOCK CONTAINS 20
                                                                                 CHARACTERS, RECORD CONTAINS 20
                                                                                 CHARACTERS, LABEL RECORDS STANDARD
    20  01  RECORD-1                           FS  000000    20  GROUP  .D005454
    21  02  FIELD-A                           FS  000000    20  AN     .D0054A8
    23  01  FILLER                            WS  000000    56  GROUP  .D0054FC
--> 24  02  KOUNT                             WS  000000     2  PACKED .D00554E <--
    25  02  ALPHABET                          WS  000002    26  AN     .D0055B2  VALUE
    26  02  ALPHA                             WS  000002     1  AN     .D00562E  REDEFINES .D0055B2, DIMENSION(26)
    28  02  NUMBR                             WS  000028     2  PACKED .D00568E
    29  02  DEPENDENTS                        WS  000030    26  AN     .D0056F2  VALUE
    30  02  DEPEND                            WS  000030     1  AN     .D005770  REDEFINES .D0056F2, DIMENSION(26)
    32  01  WORK-RECORD                       WS  000000    19  GROUP  .D0057D0
    33  02  NAME-FIELD                        WS  000000     1  AN     .D005828
    34  02  FILLER                            WS  000001     1  AN     .D00587E  VALUE
    35  02  RECORD-NO                         WS  000002     3  ZONED  .D0058D8
    36  02  FILLER                            WS  000005     1  AN     .D005940  VALUE
    37  02  LOCATION                          WS  000006     3  A      .D00599A  VALUE
    38  02  FILLER                            WS  000009     1  AN     .D005A00  VALUE
    39  02  NO-OF-DEPENDENTS                  WS  000010     2  AN     .D005A5A
    41  02  FILLER                            WS  000012     7  AN     .D005AB6  VALUE
FILE SECTION uses 20 bytes of storage
WORKING-STORAGE SECTION uses 75 bytes of storage
                  * * * * *  E N D  O F  D A T A  D I V I S I O N  M A P  * * * * *
```

Figure 18. Data Division Map for TESTPRT

Next use the IRP cross-reference listing to find the Object Definition Table (ODT) number for the internal-name. (See “Create COBOL Program Command” on page 37 and “PROCESS Statement” on page 46 for information on how to obtain these listings.) Figure 18 shows the Data Division map and Figure 19 on page 68 shows the IRP cross-reference listing for the example program, TESTPRT.

```
5728SS1                                                    IRP LISTING FOR TESTPRT
    ODT  ODT Name                          SEQ Cross Reference       (* Indicates Where Defined)
    01D8 .D005AB6 593*
    01D6 .D005A00 591*
    01D7 .D005A5A 592* 641 727 729
    01C6 .D0054A8 575*
    01C7 .D0054FC 576* 577 578 581 582
    01C5 .D005454 574* 663 717
    01C9 .D0055B2 578* 579
--> 01C8 .D00554E 577* 628 632 635 639 676 <---- 01C8 is the ODT number associated with KOUNT
    01CD .D0056F2 582* 583
    01CB .D00562E 580* 637
    01CC .D00568E 581* 629 633 643
    01D0 .D0057D0 585* 586 587 588 589 590 591 592 593 652 663 717
    01CF .D005770 584* 641
    01D3 .D0058D8 588* 643
    01D1 .D005828 586* 637
    01D2 .D00587E 587*
    01D4 .D005940 589*
    01D5 .D00599A 590*
    0254 .EXCALLX 1007 1011*
    0242 .EXCODE  966*
    0251 .EXEOP   1001* 1004
    0253 .EXFSGN  1006* 1034
    0247 .EXFS10  972* 979
    024D .EXFS22  993* 995
    024F .EXFS34  997* 999
    025B .EXGNUSE 1013 1031*
    0259 .EXIGN   1028* 1030
    0243 .EXMSGID 967* 1013
    024A .EXNOF   981* 991
    024B .EXOPEOF 982 988*
    0245 .EXOPLST 969* 985 1012
    0241 .EXPARMS 965* 966 967 968 984 1011
    0244 .EXPTR   968* 969
    0256 .EXSI    1017* 1026
    0093 .FCEXCM  188* 189
    0092 .FCLIST  187* 192
    00E9 .FCLPP   316*
    00E7 .FCLSTC  314* 315 316
    00E6 .FCLSTC# 313*
    00EC .FCLSTP  320* 321 322 323 324
    00EB .FCLSTP# 319*
    0090 .FCPARM  185* 186
    0091 .FCPARMP 186* 187
    008F .FCPTR   184* 192
    009B .FIB     206*
    00CD .FIB#OPT 285* 286 287 288 289 290 294 295 296 297
    00DB .FIB#OP1 299* 300 301 302 303 304 305
    00EA .FIBACQ  317* 1007 1009
    00E2 .FIBACTL 307* 308
    009D .FIBALT  208* 617 772 774 796 989
    00BE .FIBCA   270*
    00C1 .FIBCFMT 273*
    00A1 .FIBCFS  228* 668 669 762 764 768 780 802 847 849 863 920 975 977 983 993 997 1006 1008 1014
    00A2 .FIBCFS1 229* 716 719 883 939
    00B7 .FIBCFS2 262*
    00B3 .FIBCHAN 258*
    00BF .FIBCKID 271*
    00A0 .FIBCOP  227* 846 867 886 904 906 919 942 955
```

Figure 19. IRP Cross-Reference Listing for TESTPRT

You can use the following CL commands to add a breakpoint to the example program at statement 52 that will display the variable KOUNT, using the appropriate ODT number:

```
ENTDBG     TESTPRT
ADDBKP     STMT(52)
           PGMVAR('/01C8')
```

This command is explained in the *CL Reference.*

The following is displayed as a result of reaching this breakpoint:

```
                            Display Breakpoint

 Statement/Instruction . . . . . . . . . : 52 /0049
 Program . . . . . . . . . . . . . . . . : TESTPRT
 Recursion level . . . . . . . . . . . . : 1
 Start position  . . . . . . . . . . . . : 1
 Format  . . . . . . . . . . . . . . . . : *CHAR
 Length  . . . . . . . . . . . . . . . . : *DCL

 Variable  . . . . . . . . . . . . . . . : /01C8
   Type  . . . . . . . . . . . . . . . . :  PACKED
   Length  . . . . . . . . . . . . . . . :  2 0
  ' 26'









Press Enter to continue.

F3=Exit program   F10=Command entry
```

At this point you could change the value of program variables to alter the way in which your program is run. You can use the Change Program Variable (CHGPGMVAR) command to change the value of a variable. See the *CL Reference* for information about this command.

Considerations for Using Breakpoints

The following characteristics of breakpoints should be known before breakpoints are used:

- If a breakpoint is bypassed by a COBOL statement, for example the GO TO statement, that breakpoint is not processed.
- When a breakpoint is established on a statement, the breakpoint occurs before that statement is processed.
- Breakpoint functions are specified through OS/400 commands.

  These functions include adding breakpoints to programs, displaying breakpoint information, removing breakpoints from programs and resuming running a program after a breakpoint has been displayed. See the *CL Reference* for descriptions of these commands, and see the *CL Programmer's Guide* for a further description of breakpoints.

The following chart lists breakpoint names and the COBOL functions that they handle. These breakpoint names can appear in error messages may help you relate the statement number (field) to the function being performed.

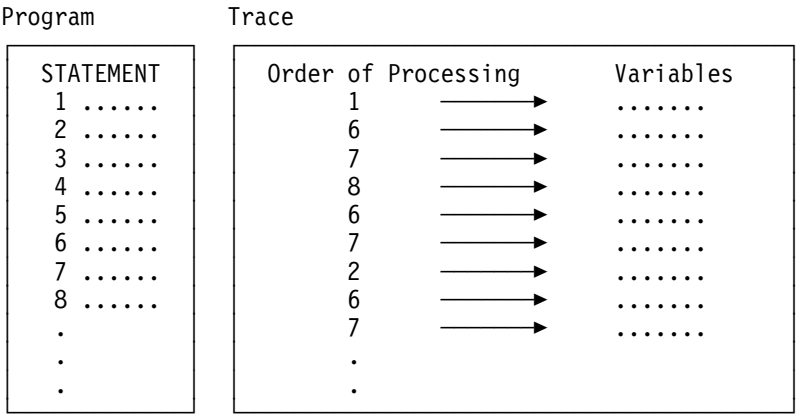
Breakpoint Name	COBOL Function
.CALEXCP	ON OVERFLOW exception monitor
.CLOSE	Close file
.CNLEXCP	CANCEL exception monitor
.CPF5004	Page overflow exception monitor
.CTRL	Control-Area processing
.DBUGBLD	Debug processing table build routine
.DBUGOUT	Debug table maintenance code
.DELETE	Delete routine
.EXCKRD	Called before doing a delete operation or a rewrite operation
.EXFSGN	Generic exception monitor
.EXFSnn	Exception monitor for file status nn
.EXINVT	Device was acquired but not invited
.EXISLD	Called before doing a sequential PUT to an indexed file
.EXNOF	File or member not found exception monitor
.EXRWRT	Called before doing a rewrite operation to an indexed file with sequential access
.EXSB23	Subfile record not found exception monitor
.EXSB24	Subfile boundary violation exception monitor
.FCEXCP	Function check exception monitor
.FEOV	Force end of volume
.GET	GET
.GETKEY	Reads the record by the key value
.GETREL	GET relative
.INIT	Initialization code and declares
.INIT002	Initialize return pointer
.LSKA	Linage control: before advancing page
.LSKB	Linage control: after advancing page
.LSPA	Linage control: before advancing n lines
.LSPB	Linage control: after advancing n lines
.LSTRT	Linage initialization
.NOALTR	Unaltered GOTO
.OPEN	Open file
.PAGING	Paging routine
.PRINIT	Printer initialization
.PRPUT	Printer PUT
.PSPA	Print control: before advancing
.PSPB	Print control: after advancing

Breakpoint Name	COBOL Function
.PUTALL	PUT
.PUTGET	PUTGET
.PUTKEY	Indexed keyed PUT
.REWRIT	Rewrite
.SIZEXCP	Internal size error exception monitor
.STARTR	Relative I/O start
.STPEXCP	STOP RUN exception monitor
.USETEST	Save the CFIB address

See the *CL Programmer's Guide* for further information on breakpoints.

Using a Trace

A trace is a record of some or all of the statements in a program that were processed. The trace can, if requested, contain the values of specific variables used in the statements.



A trace differs from a breakpoint in that a trace ends depending on which statements and how many statements are traced. The system records the traced statements that were processed. You must request a display of the traced information. The display shows the sequence in which the statements were processed and, if requested, the values of variables used in the statements.

You specify what statements the system should trace. Also, you might specify that variables be displayed only when their value changes from the previous time a traced statement was processed.

You can specify a trace of one statement in a program, a group of statements in a program, or an entire program.

Example of Using a Trace

Figure 20 shows a portion of an example COBOL program, TESTPRT. The following CL command adds a trace of statements 54 through 58 in that program. The variable NO-OF-DEPENDENTS is to be recorded only if its value changes.

```
ADDTRC          STMT(54 58)
                PGMVAR('NO-OF-DEPENDENTS') OUTVAR(*CHG)
```

Note: ENTDBG must be entered before the ADDTRC statement.

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   42  004600 PROCEDURE DIVISION.
       004700 STEP-1.
   43  004800     OPEN OUTPUT FILE-1.
   44  004900     MOVE ZERO TO KOUNT, NUMBR.
       005000*******************************************************
       005100* THE FOLLOWING 3 PARAGRAPHS CREATE INTERNALLY THE  *
       005200* RECORDS TO BE CONTAINED IN THE FILE, WRITES THEM  *
       005300* ON THE DISK, AND DISPLAYS THEM                    *
       005400*******************************************************
       005500 STEP-2.
   45  005600     ADD 1 TO KOUNT, NUMBR.
   46  005700     MOVE ALPHA  (KOUNT) TO NAME-FIELD.
   47  005800     MOVE DEPEND (KOUNT) TO NO-OF-DEPENDENTS.
   48  005900     MOVE NUMBR          TO RECORD-NO.
       006000 STEP-3.
   49  006100     DISPLAY WORK-RECORD.
   50  006200     WRITE RECORD-1 FROM WORK-RECORD.
       006300 STEP-4.
   51  006400     PERFORM STEP-2 THRU STEP-3 UNTIL KOUNT IS EQUAL TO 26.
       006500*******************************************************
       006600* THE FOLLOWING PARAGRAPH CLOSES FILE OPENED FOR    *
       006700* OUTPUT AND RE-OPENS IT FOR INPUT                  *
       006800*******************************************************
       006900 STEP-5.
   52  007000     CLOSE FILE-1.
   53  007100     OPEN INPUT FILE-1.
       007200*******************************************************
       007300* THE FOLLOWING PARAGRAPHS READS BACK THE FILE AND  *
       007400* SINGLES OUT EMPLOYEES WITH NO DEPENDENTS          *
       007500*******************************************************
       007600 STEP-6.
-->54  007700     READ FILE-1 RECORD INTO WORK-RECORD
   55  007800       AT END GO TO STEP-8.
       007900 STEP-7.
   56  008000     IF NO-OF-DEPENDENTS IS EQUAL TO "0"
   57  008100       MOVE "Z" TO NO-OF-DEPENDENTS.
-->58  008200     GO TO STEP-6.
       008300 STEP-8.
   59  008400     CLOSE FILE-1.
   60  008500     STOP RUN.
```

Figure 20. COBOL Source Code

Figure 21 on page 73 is an example of a display of the traced information. The following CL command line lists this information:

```
DSPTRCDTA OUTPUT(*LIST) CLEAR(*YES)
```

The Display Trace Data command is explained in the *IBM System/38 Control Language Reference Manual*.

```
5728SS1                              Display Trace Data
                Statement/
Program         Instruction           Recursion Level       Sequence Number
TESTPRT         54                           1                          1
 Start position  . . . . . . . . . . . . : 1
 Length  . . . . . . . . . . . . . . . . : *DCL
 Format  . . . . . . . . . . . . . . . . : *CHAR
 Variable  . . . . . . . . . . . . . . . : 05  NO-OF-DEPENDENTS
   Type  . . . . . . . . . . . . . . . . :   CHARACTER
   Length  . . . . . . . . . . . . . . . :   2
   *...+....1....+....2....+....3....+....4....+....5
  '0 '
                Statement/
Program         Instruction           Recursion Level       Sequence Number
TESTPRT         56                           1                          2
TESTPRT         57                           1                          3
TESTPRT         58                           1                          4
 Start position  . . . . . . . . . . . . : 1
 Length  . . . . . . . . . . . . . . . . : *DCL
 Format  . . . . . . . . . . . . . . . . : *CHAR
*Variable  . . . . . . . . . . . . . . . : 05  NO-OF-DEPENDENTS
   Type  . . . . . . . . . . . . . . . . :   CHARACTER
   Length  . . . . . . . . . . . . . . . :   2
   *...+....1....+....2....+....3....+....4....+....5
  'Z '
                Statement/
Program         Instruction           Recursion Level       Sequence Number
TESTPRT         54                           1                          5
TESTPRT         56                           1                          6
 Start position  . . . . . . . . . . . . : 1
 Length  . . . . . . . . . . . . . . . . : *DCL
 Format  . . . . . . . . . . . . . . . . : *CHAR
*Variable  . . . . . . . . . . . . . . . : 05  NO-OF-DEPENDENTS
   Type  . . . . . . . . . . . . . . . . :   CHARACTER
   Length  . . . . . . . . . . . . . . . :   2
   *...+....1....+....2....+....3....+....4....+....5
  '1 '
                Statement/
Program         Instruction           Recursion Level       Sequence Number
TESTPRT         58                           1                          7
TESTPRT         54                           1                          8
TESTPRT         56                           1                          9
 Start position  . . . . . . . . . . . . : 1
 Length  . . . . . . . . . . . . . . . . : *DCL
 Format  . . . . . . . . . . . . . . . . : *CHAR
*Variable  . . . . . . . . . . . . . . . : 05  NO-OF-DEPENDENTS
   Type  . . . . . . . . . . . . . . . . :   CHARACTER
   Length  . . . . . . . . . . . . . . . :   2
   *...+....1....+....2....+....3....+....4....+....5
  '2 '
```

Figure 21. Trace Data Display Listing

Considerations for Using a Trace

The following characteristics of traces should be known before traces are used in COBOL programs:

- Statements bypassed by COBOL statements, such as the `GO TO` statement, are not included in the trace.
- Trace functions are specified through CL commands in the job that contains the traced program. These functions include adding trace requests to a program, removing trace requests from a program, removing data collected from previous traces, displaying trace information, and displaying the traces that have been specified for a program.

- In addition to statement numbers, names of COBOL-generated routines can appear on the trace output STMT field.

See the *CL Programmer's Guide* for a further description of traces.

Using a Debug Run-Time Switch

A run-time switch is provided for the COBOL Debug facility. This switch activates the debugging code generated when WITH DEBUGGING MODE is specified. When the run-time switch is set on, all debugging sections that were compiled are activated. When the run-time switch is set off (the default), the USE FOR DEBUGGING Declarative procedures are inhibited. In both cases the debugging lines (D in column 7) remain in effect.

See "DEBUGGING FEATURES" on page 517 in Chapter 11, "Using the Additional COBOL Functions" for more information on COBOL Debug and how to use the switch.

File Status

The following format can be used to transfer data (OPEN-FEEDBACK or I-O-FEEDBACK areas) associated with an open file to an identifier.

Format

```
ACCEPT identifier FROM mnemonic-name
  [ FOR file-name ]
```

See Chapter 10, "Procedure Division" for more information on specifying this statement.

Appendix E, "File Structure Support Summary and Status Key Values" contains a discussion of the OPEN-FEEDBACK and I-O-FEEDBACK areas. See the *System/38 Environment Programmer's Guide/Reference* for a layout and description of the data areas contained in the feedback areas.

Note: Certain values can only occur on a System/38 and are not applicable to the AS/400 system.

Using a COBOL Formatted Dump

Some COBOL run-time messages allow you to obtain a COBOL formatted dump option by giving a reply of D or F.

The output for the dump is sent to the IBM-supplied printer file QPPGMDMP.

The formatted dump (reply D) includes the current file information about the files in the program, contents of fields, data structures, arrays, and tables for user-defined COBOL data variables.

If you reply with an F option, the dump also includes a list of compiler-generated fields and their contents.

Both the D option and the F option will dump the first 256 characters of program variables. Any variable greater than 256 characters will be truncated.

If you do not desire a dump, specify reply C (cancel with *no* dump). Reply C is also the default reply for all COBOL inquiry messages that allow a dump to be obtained.

Reply Modes and System Reply List

The need for a reply to run-time inquiry messages is controlled by the INQMSGRPY parameter on the following control language (CL) commands:

- CHGJOB — Change Job
- CHGJOBD — Change Job Description
- CRTJOBD — Create Job Description
- JOB — Job
- SBMJOB — Submit Job.

Through the INQMSGRPY parameter, three reply modes for inquiry messages are possible. The reply modes are:

- *RQD — A reply is required.
- *DFT — Take the default reply.
- *SYSRPL — Search the System Reply List and if the message is found, take the default action specified in the list. If the message is not found in the System Reply List, then take the default specified in the message description.

The System Reply List allows you to specify replies for inquiry messages CBE7200, CBE7201, CBE7203, CBE7204, and CBE7205. The replies may be specified individually or generically. This method of replying to inquiry messages is especially suitable for batch programs, which would otherwise require a console operator to issue replies.

To see the entries in the System Reply List, specify the Display Reply List CL command (DSPRPYL). This allows you to change, remove, and add entries to the System Reply List.

For more information on message reply modes and the System Reply List, see the *CL Programmer's Guide.*

Example of Using a Dump

Figure 22 on page 77 shows an example of a COBOL formatted dump.

The following list describes the areas of Figure 22 on page 77 indicated by letters:

A The exception for which the dump was requested and the location in the program where the exception occurred.

B The COBOL statement number of the last I-O operation that was processed before the exception occurred. This information is produced only if at least one I-O operation has been processed.

C Current information for each file. This information is produced only if the program has files.

D Beginning of compiler-generated fields (included in the dump if the user responds with an F option).

E I-O flags for the current file:

Bit	Meaning
1	File is open
2	File is locked
3	End of file
4	(Not Used)
5	Optional file
6	Check indexed file for duplicates at open
7	End of page
8	(Reserved)

F Previous status code.

G Beginning of Module Global Table (MGT).

H Last exception code.

I Call level of current program.

J Qualified program name and library.

K Beginning of the Program Global Table (PGT).

L Call level of the main COBOL program.

M Job date (YYMMDD).

N Beginning of user fields.

O Invalid zoned field printed in hexadecimal.

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.     EXMPLEDUMP.
    3  000300   AUTHOR.       PROGRAMMER NAME.
    4  000400   INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
    5  000500   DATE-WRITTEN. 08/26/88.
    6  000600   DATE-COMPILED.  08/26/88 15:00:18   .
    7  000700 ENVIRONMENT DIVISION.
    8  000800 CONFIGURATION SECTION.
    9  000900 SOURCE-COMPUTER. IBM-S38.
   10  001000 OBJECT-COMPUTER. IBM-S38.
   11  001100 INPUT-OUTPUT SECTION.
   12  001200 FILE-CONTROL.
   13  001300     SELECT FILE-1 ASSIGN TO DISK-SALESFILE.
   14  001400 DATA DIVISION.
   15  001500 FILE SECTION.
   16  001600 FD  FILE-1
   17  001700     LABEL RECORDS ARE STANDARD.
   18  001800 01  RECORD-1.
   19  001900   05 R-TYPE             PIC X(1).
   20  002000   05 R-AREA-CODE        PIC 9(2).
   21  002100     88 R-NORTH-EAST VALUES 15 THROUGH 30.
   22  002200   05 R-SALES-CAT-1      PIC S9(5)V9(2) COMP-3.
   23  002300   05 R-SALES-CAT-2      PIC S9(5)V9(2) COMP-3.
   24  002400   05 FILLER             PIC X(1).
   25  002500
   26  002600 WORKING-STORAGE SECTION.
   27  002700 01  W-SALES-VALUES.
   28  002800     05 W-CAT-1          PIC S9(8)V9(2).
   29  002900     05 W-CAT-2          PIC S9(8)V9(2).
   30  003000     05 W-TOTAL          PIC S9(8)V9(2).
   31  003100
   32  003200 01  W-EDIT-VALUES.
   33  003300     05 FILLER           PIC X(8) VALUE "TOTALS: ".
   34  003400     05 W-EDIT-1         PIC Z(7)9.9(2)-.
   35  003500     05 FILLER           PIC X(3) VALUE SPACES.
   36  003600     05 W-EDIT-2         PIC Z(7)9.9(2)-.
   37  003700     05 FILLER           PIC X(3) VALUE SPACES.
   38  003800     05 W-EDIT-TOTAL     PIC Z(7)9.9(2)-.
   39  003900
   40  004000 01  END-FLAG           PIC X(1) VALUE SPACE.
   41  004100     88 END-OF-INPUT VALUE "Y".
   42  004200
   43  004300 PROCEDURE DIVISION.
       004400*****************************************************************
       004500* OPEN THE INPUT FILE, CLEAR TOTALS, CALL MAIN PROCESS THEN    *
       004600* DISPLAY THE RESULTS AND END THE RUN.                         *
       004700*****************************************************************
       004800 P-START.
   44  004900     OPEN INPUT FILE-1.
   45  005000     MOVE ZEROS TO W-SALES-VALUES.
   46  005100     PERFORM P-MAIN UNTIL END-OF-INPUT.
       005200
   47  005300     MOVE W-CAT-1 TO W-EDIT-1.
   48  005400     MOVE W-CAT-2 TO W-EDIT-2.
   49  005500     MOVE W-TOTAL TO W-EDIT-TOTAL.


5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   50  005600     DISPLAY W-EDIT-VALUES.
   51  005700     STOP RUN.
       005800
       005900*****************************************************************
       006000* READ THE INPUT FILE PROCESSING ONLY THOSE RECORDS FOR THE    *
       006100* NORTH EAST AREA. WHEN END-OF-INPUT REACHED, SET THE FLAG     *
       006200*****************************************************************
       006300 P-MAIN.
   52  006400     READ FILE-1 AT END SET END-OF-INPUT TO TRUE.
   54  006500     IF R-NORTH-EAST AND NOT END-OF-INPUT
   55  006600         ADD R-SALES-CAT-1 TO W-CAT-1, W-TOTAL
   56  006700         ADD R-SALES-CAT-2 TO W-CAT-2, W-TOTAL.
                        * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 22 (Part 1 of 10). Example of a COBOL Formatted Dump

```
 A
MCH1202 exception in program EXMPLEDUMP in QTEMP at MI instruction number 0049 COBOL statement number 55.
 B
Last I-O operation was at statement 52.
 C


CBE7903-Information pertaining to file FILE-1.
CBE7905-File is open.
CBE7906-Last I-O operation completed for file was READ.
CBE7907-Last file status for file was 00.
CBE7910-Last extended file status for file was     .
FORMATTED DATA DUMP FOR PROGRAM EXMPLEDUMP.QTEMP
NAME           OFFSET  ATTRIBUTES    VALUE  D
.ADBUF         000590  POINTER(SPP)  NULL
.ADBUFVL       000C90  CHAR(68)      '                                                                    '
               000C90  VALUE IN HEX  '00000000000000000000000000000000000000000000000000000000000000000000000000000000'X
               000CB8    +41         '00000000000000000000000000000000000000000000000000000000'X
.ADDEV         0005C4  CHAR(10)      '          '                    '0000000000000000000'X
.ADENV         0005A3  CHAR(1)       'I'
.ADFILE        0005D0  POINTER(SPP)  NULL
.ADFUNC        0005A0  CHAR(1)       ' '                            '00'X
.ADLN          0005A4  BINARY(2)     0
.ADMID         0005A6  BINARY(2)     0
.ADPGM         0005AB  CHAR(10)      'EXMPLEDUMP'
.ADRLN         0005A8  BINARY(2)     0
.ADRTN         000580  POINTER(IP)   NULL
.ADRTYP        0005AA  CHAR(1)       ' '                            '00'X
.ADTOD         0005B5  CHAR(15)      '               '              '000000000000000000000000000'X
.ADTYP         0005A1  BINARY(2)     0
.BINSUB        000688  BINARY(2)     0
.BIN2          00068A  BINARY(2)     0
.BPCA          0004B0  CHAR(32767)   '            ¢                  D01   <HHH              D12   <HHH              D15   <HHH'
               00050A    +91         '              D15   <HHH             D22   <HHH              D99   <HHH              D01 '
               000564    +181        '  <HHH             D23   <HHH             D25   <HHH             D88   <H'
               0004B0  VALUE IN HEX  '8000000000000000026C0194A000A00000100AA0018000E4040404040404040C4F0F14444444C88'X
               0004D8    +41         '88888C4040400001000000010000000C4F1F24444444C8888888C4040400001000000020000000'X
               000500    +81         'C4F1F54444444C8888888C4040400001000000030000000C4F1F54444444C8888888C4040400001'X
               000528    +121        '00000004000000000C4F2F24444444C8888888C4040400001000000050000000C4F9F94444444C88'X
               000550    +161        '88888C4040400001000000060000000C4F0F14444444C8888888C4040400001000000070000000'X
               000578    +201        'C4F2F34444444C8888888C4040400001000000080000000C4F2F54444444C8888888C4040400001'X
               0005A0    +241        '0000000900000000C4F8F84444444C88'X
.BPCACTR       0004C0  BINARY(2)     1
.BPCAMXR       0004C2  BINARY(2)     170
.BPCARCD       0004B0  POINTER(SPP)  SPACE OFFSET    1616     '00000650'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.BPCARIO       0004C4  BINARY(2)     24
.BP01CA        0004B0  CHAR(32767)   '            ¢                  D01   <HHH              D12   <HHH              D15   <HHH'
               00050A    +91         '              D15   <HHH             D22   <HHH              D99   <HHH              D01 '
               000564    +181        '  <HHH             D23   <HHH             D25   <HHH             D88   <H'
               0004B0  VALUE IN HEX  '8000000000000000026C0194A000A00000100AA0018000E4040404040404040C4F0F14444444C88'X
               0004D8    +41         '88888C4040400001000000010000000C4F1F24444444C8888888C4040400001000000020000000'X
               000500    +81         'C4F1F54444444C8888888C4040400001000000030000000C4F1F54444444C8888888C4040400001'X
               000528    +121        '00000004000000000C4F2F24444444C8888888C4040400001000000050000000C4F9F94444444C88'X
               000550    +161        '88888C4040400001000000060000000C4F0F14444444C8888888C4040400001000000070000000'X
               000578    +201        'C4F2F34444444C8888888C4040400001000000080000000C4F2F54444444C8888888C4040400001'X
               0005A0    +241        '0000000900000000C4F8F84444444C88'X
.BP01CTR       0004C0  BINARY(2)     1
.BP01MXR       0004C2  BINARY(2)     170
.BP01RCD       0004B0  POINTER(SPP)  SPACE OFFSET    1616     '00000650'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.BP01RIO       0004C4  BINARY(2)     24
.BSTRING               NOT ADDRESSABLE
.BUFFER                NOT ADDRESSABLE
.BUFPTR        0008B0  POINTER(SPP)  NULL
.CALERP        0006C0  POINTER(SPP)  SPACE OFFSET    1680     '00000690'X
                                     OBJECT    PSSA
.CALPHAB       000D20  CHAR(27)      ' ETAOINSHRDLUCMFWYPVBGKQJXZ'
.CIMBSGN       000CDA  CHAR(60)      '0123456789 JKLMNOPQR  STUVWXYZ           ABCDEFGHI  STUVWXYZ'
               000CDA  VALUE IN HEX  'F0F1F2F3F4F5F6F7F8F9D0D1D2D3D4D5D6D7D8D9A0A1A2A3A4A5A6A7A8A9B0B1B2B3B4B5B6B7B8B9'X
```

Figure 22 (Part 2 of 10). Example of a COBOL Formatted Dump

```
FORMATTED DATA DUMP FOR PROGRAM EXMPLEDUMP.QTEMP
NAME           OFFSET  ATTRIBUTES    VALUE
               000D02    +41         'C0C1C2C3C4C5C6C7C8C9E0E1E2E3E4E5E6E7E8E9'X
.CNLERP        0006B0  POINTER(SPP)  SPACE OFFSET    1680     '00000690'X
                                     OBJECT    PSSA
.CNUMERC       000D16  CHAR(10)      '0123456789'
.CPADCHR       000D3B  CHAR(1)       ' '
.CRCLEAR       000DB0  POINTER(SYP)  OBJECT    QCRCLEAR
                                     CONTEXT   QSYS
.CSEPSGN       000CD8  CHAR(2)       '+-'
.DBUGRTN       000560  POINTER(IP)   NULL
.DEVPTR        000870  POINTER(SPP)  SPACE OFFSET    324     '00000144'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.DISPPOS       000D60  BINARY(2)     0
.DISPPTR       000D50  POINTER(SPP)  NULL
.DLINENO       000682  CHAR(6)       '      '                        '000000000000'X
.DMCACIN       0009B0  BINARY(2)     121
.DMCACQR       0009B2  BINARY(2)     66
.DMCCPCL       000178  BINARY(2)     13
.DMCCPOP       00017A  BINARY(2)     17
.DMCDELT       000166  BINARY(2)     69
.DMCDROP       0009B4  BINARY(2)     71
.DMCFEOD       000168  BINARY(2)     111
.DMCFRCE       00016A  BINARY(2)     69
.DMCGET        000158  BINARY(2)     770
.DMCGETD       00015A  BINARY(2)     14
.DMCGETK       00015C  BINARY(2)     69
.DMCLINK       0008E5  BINARY(2)     0
.DMCODP        000000  CHAR(32767)   'E      Q   Q           F                                                    ¢             '
               00005A    +91         '                  A               ¢                                                  DBSA'
               0000B4    +181        'LESFILE COB38EX                          SALESFILE                        A'
               000000  VALUE IN HEX  '85000002000014D8000014D8000000B000000140000001C60000028000000000000002C000000000'X
               000028    +41         '00000000000001C000000000000000000000000000000000000000000000000000026C0194A000DFF'X
               000050    +81         '00000000000000000000000000000000800000000000000000026C00081000C10800000000000000'X
               000078    +121        '0026C0194A00187000000000000000000000000000000000190000000000000000000C00000000'X
               0000A0    +161        '0B000000000000000004B00000000000C4C2E2C1D3C5E2C6C9D3C540C3D6C2F3F8C5E74040400000'X
               0000C8    +201        '00000000000000000000000000000000000000000000E0000E2C1D3C5E2C6C9D3C540000010080000'X
               0000F0    +241        '000000000000000000000000000011C1'X
.DMCOFFS       000010  BINARY(4)     320
.DMCPTGT       000162  BINARY(2)     69
.DMCPUT        000160  BINARY(2)     69
.DMCPUTD       00015E  BINARY(2)     69
.DMCRLSE       000170  BINARY(2)     69
.DMCRSTD       00016E  BINARY(2)     69
.DMCSPDD       00016C  BINARY(2)     69
.DMCSPTB       00017C  BINARY(2)     0
.DMCTBLE       00017E  BINARY(2)     1
.DMCUPD        000164  BINARY(2)     69
.DMPBDSE               NOT ADDRESSABLE
.DMPCDFO       0001C6  BINARY(2)     144
.DMPCDFP       0008D0  POINTER(SPP)  NULL
.DMPDBFB               NOT ADDRESSABLE
.DMPDBFL       000190  CHAR(1)       ' '                              '00'X
.DMPDENT       000144  CHAR(130)     DIMENSION(250)
               000144    (1)         'DATABASE                             ?                                                '
               00019E    +91         '                                       '
               000144  VALUE IN HEX  'C4C1E3C1C2C1E2C5404000000000000000000000000302000E00450045004500450045006F0045'X
               00016C    +41         '00450045004500450045000D0011000000010000000000000000000000000000000000000000'X
   .
   .
   .
```

Figure 22 (Part 3 of 10). Example of a COBOL Formatted Dump

```
FORMATTED DATA DUMP FOR PROGRAM EXMPLEDUMP.QTEMP
NAME           OFFSET  ATTRIBUTES    VALUE
.FIB#OPT       00076C  CHAR(8)       '        '                    '0000000000000000'X
.FIB#OP1               NOT ADDRESSABLE
.FIBACQ        000828  BINARY(2)     0
.FIBACTL       000800  CHAR(8)       '        '                    '16000400000000FF'X
.FIBALT        0009EE  CHAR(1)       ' '                           '00'X
.FIBCA         000A85  CHAR(22)      '                      '      '00000000000000000000000000000000000000000000'X
.FIBCFMT       000A91  CHAR(10)      '          '                  '00000000000000000000'X
.FIBCFS        0009F4  CHAR(2)       '00'
.FIBCFS1       0009F4  CHAR(1)       '0'
.FIBCFS2       000A7C  CHAR(4)       '    '                        '00000000'X
.FIBCHAN       000A50  POINTER(SPP)  NULL
.FIBCKID       000A85  ZONED(2,0)    **                            '0000'X
.FIBCOP        0009F0  CHAR(4)       '    '                        '03000001'X
.FIBCTID       000A87  CHAR(10)      '          '                  '00000000000000000000'X
.FIBCTL        000890  POINTER(SPP)  NULL
.FIBCUR        0009F0  CHAR(6)       '    00'                      '03000001F0F0'X
.FIBCURK       000774  CHAR(123)     '                                                                                                '
               0007CE    +91         '                                  '
               000774  VALUE IN HEX  '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'X
               00079C    +41         3 LINES OF ZEROES SUPPRESSED
.FIBDEVI       000A7A  BINARY(2)     1
.FIBDEVN       000A70  CHAR(10)      '          '
.FIBFLGS       0009EF  CHAR(1)       ' ' E                         '80'X
.FIBFMT        000A0A  CHAR(10)      '          '
.FIBFN         0009D0  CHAR(30)      'FILE-1                      '
.FIBK#LN       00076D  BINARY(2)     0
.FIBK#R#       00076F  BINARY(4)     0
.FIBK#RK       000771  BINARY(2)     0
.FIBK#TP       00076C  CHAR(1)       ' '                           '00'X
.FIBKCGK       000740  CHAR(8)       '        '                    '0800040000000009'X
.FIBKCGR       000748  CHAR(8)       '        '                    '02000400000000FF'X
.FIBKCPD       000750  CHAR(8)       '        '                    '04000400000000FF'X
.FIBKCTL       000758  BINARY(2)     0
.FIBKDLN       000768  BINARY(2)     2
.FIBKDM#       00076A  BINARY(2)     0
.FIBKDTP       000767  CHAR(1)       ' '                           '0F'X
.FIBKEY        0009FA  BINARY(4)     0
.FIBKFLN       00075B  BINARY(2)     10
.FIBKFMT       00075D  CHAR(10)      '          '
.FIBKFTP       00075A  CHAR(1)       ' '                           '01'X
.FIBKKEY       000776  CHAR(121)     '                                                                                                '
               0007D0    +91         '                                  '
               000776  VALUE IN HEX  '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'X
               00079E    +41         3 LINES OF ZEROES SUPPRESSED
.FIBKKLN       000774  BINARY(2)     0
.FIBKKTP       000773  CHAR(1)       ' '                           '00'X
.FIBKLEN       0009FE  BINARY(2)     0
.FIBKSTC       00076C  CHAR(1)       ' '                           '00'X
.FIBKSTE       000771  CHAR(1)       ' '                           '00'X
.FIBKSTL       00076D  BINARY(2)     0
.FIBKSTT       00076F  BINARY(2)     0
.FIBLBO        000A00  BINARY(2)     0
.FIBLFT        0009FC  BINARY(2)     0
.FIBLIN        0009FA  BINARY(2)     0
.FIBLINE       000A02  BINARY(2)     0
.FIBLTO        0009FE  BINARY(2)     0
```

Figure 22 (Part 4 of 10). Example of a COBOL Formatted Dump

```
FORMATTED DATA DUMP FOR PROGRAM EXMPLEDUMP.QTEMP
NAME           OFFSET  ATTRIBUTES    VALUE
.FIBMBRN       000B06  CHAR(10)      'SALESFILE '
.FIBOFMT       000AFC  CHAR(10)      '          '                    '0000000000000000000'X
.FIBOFS        0009F6  CHAR(2)       '  ' F                         '0000'X
.FIBOKEY       000A83  CHAR(121)     '                                                                                                  '
               000ADD    +91         '                                '
               000A83  VALUE IN HEX  '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'X
               000AAB    +41         3 LINES OF ZEROES SUPPRESSED
.FIBOKLN       000A81  BINARY(2)     0
.FIBOLDK       000A81  CHAR(123)     '                                                                                                  '
               000ADB    +91         '                                  '
               000A81  VALUE IN HEX  '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'X
               000AA9    +41         3 LINES OF ZEROES SUPPRESSED
.FIBOP         0008E1  CHAR(4)       '    '                           '03000001'X
.FIBOP1        0008E1  CHAR(1)       ' '                              '03'X
.FIBOP2        0008E2  CHAR(1)       ' '                              '00'X
.FIBOP3        0008E3  CHAR(1)       ' '                              '00'X
.FIBOP4        0008E4  CHAR(1)       ' '                              '01'X
.FIBORRN       000A81  BINARY(4)     0
.FIBPTR        0004C0  POINTER(SPP)  SPACE OFFSET    2512     '000009D0'X
                                     OBJECT    PSSA
.FIBP1         000A60  POINTER(SPP)  SPACE OFFSET    1200     '000004B0'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.FIBREL                NOT ADDRESSABLE
.FIBRLPT       0007F0  POINTER(SPP)  NULL
.FIBROLC               NOT ADDRESSABLE
.FIBROLE               NOT ADDRESSABLE
.FIBROLL               NOT ADDRESSABLE
.FIBRSL                NOT ADDRESSABLE
.FIBRVAL               NOT ADDRESSABLE
.FIBSPC        0009FA  CHAR(14)      '              '                 '00000000000000000000000000'X
.FIBTAPE       000808  CHAR(8)       '        '                       '11000400000000FF'X
.FIBTLEN       00080B  BINARY(4)     0
.FIBUBTO       000A30  POINTER(IP)   NULL
.FIBUFCB       000A20  POINTER(SPP)  SPACE OFFSET    2832     '00000B10'X
                                     OBJECT    PSSA
.FIBURTN       000A40  POINTER(SPP)  NULL
.FIBUSAV       000730  POINTER(IP)   NULL
.FIBUSE#       000A08  BINARY(2)     0
.FIBVERB       0009F8  BINARY(2)     4
.FSKA          00084C  BINARY(2)     0
.FSKB          000842  BINARY(2)     0
.FSPA          000851  BINARY(2)     0
.FSPB          000847  BINARY(2)     0
.FSTKS         000825  BINARY(2)     0
.FWTRCD        000803  BINARY(4)     0
.F01ALTS       0009EE  CHAR(1)       ' '                              '00'X
.F01CFS2       000A7C  CHAR(4)       '    '                           '00000000'X
.F01CHAN       000A50  POINTER(SPP)  NULL
.F01COP        0009F0  CHAR(4)       '    '                           '03000001'X
.F01CUR        0009F0  CHAR(6)       '    00'                         '03000001F0F0'X
.F01DEVI       000A7A  BINARY(2)     1
.F01DEVN       000A70  CHAR(10)      '          '
.F01FLGS       0009EF  CHAR(1)       ' '                              '80'X
.F01FMT        000A0A  CHAR(10)      '          '
.F01FN         0009D0  CHAR(30)      'FILE-1                        '
.F01MBRN       000B06  CHAR(10)      'SALESFILE '
```

Figure 22 (Part 5 of 10). Example of a COBOL Formatted Dump

```
FORMATTED DATA DUMP FOR PROGRAM EXMPLEDUMP.QTEMP
NAME           OFFSET  ATTRIBUTES    VALUE
.F01OFMT       000AFC  CHAR(10)      '          '                    '00000000000000000000'X
.F01OFS        0009F6  CHAR(2)       '  '                           '0000'X
.F01OKLN       000A81  BINARY(2)     0
.F01OLDK       000A83  CHAR(121)     '                                                                                                  '
               000ADD    +91         '                                '
               000A83  VALUE IN HEX  '0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'X
               000AAB    +41         3 LINES OF ZEROES SUPPRESSED
.F01P1         000A60  POINTER(SPP)  SPACE OFFSET    1200     '000004B0'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.F01SPC        0009FA  CHAR(14)      '              '               '00000000000000000000000000'X
.F01UBTO       000A30  POINTER(IP)   NULL
.F01UFCB       000A20  POINTER(SPP)  SPACE OFFSET    2832     '00000B10'X
                                     OBJECT    PSSA
.F01URTN       000A40  POINTER(SPP)  NULL
.F01USE#       000A08  BINARY(2)     0
.F01VERB       0009F8  CHAR(2)       '  '                           '0004'X
.IOCPTR        0009A0  POINTER(SPP)  SPACE OFFSET    2272     '000008E0'X
                                     OBJECT    PSSA
.IOEPTR        000980  POINTER(SYP)  OBJECT    QDBGETM
                                     CONTEXT   QSYS
.IOFDBEX       0008C0  POINTER(SPP)  NULL
.IOOPTR        000990  POINTER(SPP)  SPACE OFFSET    2544     '000009F0'X
                                     OBJECT    PSSA
.IORTN         0004B0  POINTER(IP)   STMT 52                          INSTR # 00000041
                                     OBJECT    EXMPLEDUMP
                                     CONTEXT   QTEMP
.IP00001       000D40  POINTER(IP)   STMT 56                          INSTR # 0000004E
                                     OBJECT    EXMPLEDUMP
                                     CONTEXT   QTEMP
.MAINRTN       000550  POINTER(SYP)  OBJECT    QCRMAIN
                                     CONTEXT   QSYS
.MGT           000340  CHAR(16)      'COBOL MGT 00.0  ' G
.MGTBIN8       000457  CHAR(8)       '        '                      '0000000000000000'X
.MGTCNCL       000436  CHAR(1)       '0'
.MGTCNTR       0003D0  BINARY(4)     DIMENSION(20)
               00041C    (40-20)     0
.MGTCPGM       0004A0  POINTER(SYP)  NULL
.MGTDBUG       000438  CHAR(1)       '0'
.MGTEXCP       00042C  CHAR(7)       '       ' H
.MGTFIB        000360  POINTER(SPP)  SPACE OFFSET    2512     '000009D0'X
                                     OBJECT    PSSA
.MGTFUNC       000455  BINARY(2)     2
.MGTIND        000433  CHAR(1)       DIMENSION(32)
               000439    (1-7)       '0'
               00043A    (8)         '1'
               000452    (9-32)      '0'
.MGTINVC       00042A  BINARY(2)     5 I
.MGTLIB        00052A  CHAR(10)      '          '                    '00000000000000000000'X
.MGTMSGI       000502  CHAR(7)       '       '                       '00000000000000'X
.MGTMSGN       000500  BINARY(2)     0
.MGTMSGR       0004F0  POINTER(SPP)  NULL
.MGTMSGS       0004D0  POINTER(IP)   NULL
.MGTMSGT       0004E0  POINTER(SPP)  NULL
.MGTNAME       000420  CHAR(10)      'EXMPLEDUMP'
.MGTNEXT       000350  POINTER(SPP)  NULL
.MGTOSZ        000433  CHAR(1)       '0'
```

Figure 22 (Part 6 of 10). Example of a COBOL Formatted Dump

```
FORMATTED DATA DUMP FOR PROGRAM EXMPLEDUMP.QTEMP
NAME           OFFSET  ATTRIBUTES    VALUE
.MGTOVFL       000435  CHAR(1)       '0'
.MGTPACK       00045F  PACKED(31,0)  ******************************** '0000000000000000000000000000000C'X
.MGTPARM       000510  POINTER(SPP)  NULL
.MGTPASA       000380  POINTER(SPP)  SPACE OFFSET    11728     '00002DD0'X
                                     OBJECT    PASA
.MGTPASC       000380  CHAR(16)      '            B   '              '80000000000000000026C00082002ED0'X
.MGTPCS        000480  POINTER(SPP)  NULL
.MGTPFM        000437  CHAR(1)       '0'
.MGTPGM        0003B0  POINTER(SYP)  OBJECT    EXMPLEDUMP  J
                                     CONTEXT   QTEMP
.MGTPGT        000370  POINTER(SPP)  SPACE OFFSET    6048     '000017A0'X
                                     OBJECT    PSSA
.MGTPLVL       000471  BINARY(2)     0
.MGTPROG       000520  CHAR(10)      '          '                    '00000000000000000000'X
.MGTPTP        000490  POINTER(SPP)  SPACE OFFSET    3152     '00000C50'X
                                     OBJECT    PSSA
.MGTPTR        000570  POINTER(SPP)  SPACE OFFSET    832     '00000340'X
                                     OBJECT    PSSA
.MGTRST        0003C0  POINTER(IP)   NULL
.MGTSEG        00046F  BINARY(2)     0
.MGTSEPT       000390  POINTER(SPP)  SPACE OFFSET    0     '00000000'X
                                     OBJECT    QINSEPT
                                     CONTEXT   QSYS
.MGTSOSZ       000434  CHAR(1)       '0'
.MGTSPCD       000439  CHAR(1)       '0'
.MGTSW         000453  CHAR(1)       ' '                              '80'X
.MGTTYPE       000454  CHAR(1)       'I'
.MGTUPTR       0003A0  POINTER(SPP)  SPACE OFFSET    1984     '000007C0'X
                                     OBJECT    E41       PARIS     007517
.MGT9001       00043A  CHAR(1)       '1'
.NULLCL        0008E0  CHAR(1)       ' '                              'FF'X
.ODPBPTR       0008A0  POINTER(SPP)  SPACE OFFSET    0     '00000000'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.ONSAVE        0005E0  CHAR(32)      '                                  '
               0005E0  VALUE IN HEX  '00000000000000000000000000000000000000000000000000000000000000000000000000000000'X
.ONSAV1        000600  CHAR(32)      '                                  '
               000600  VALUE IN HEX  '00000000000000000000000000000000000000000000000000000000000000000000000000000000'X
.PBPDUM        0009D0  POINTER(IP)   NULL
.PBP0003       000C60  POINTER(IP)   STMT 46                          INSTR # 00000027
                                     OBJECT    EXMPLEDUMP
                                     CONTEXT   QTEMP
.PCB                   NOT ADDRESSABLE
.PCBAR                 NOT ADDRESSABLE
.PCBLIB                NOT ADDRESSABLE
.PCBPROG               NOT ADDRESSABLE
.PCBPTR        0009C0  POINTER(SPP)  NULL
.PERFCTR       000680  BINARY(2)     1
                                       K
.PGT           0017A0  CHAR(32767)   'PGT 00.0                    A                  01000000000000000000000000000000     (    '
               0017FA    +91         '                                                                                .              '
               001854    +181        '                                                                                    '
               0017A0  VALUE IN HEX  'D7C7E340F0F04BF0404040404040404080000000000000000026C000810004400000000000000000'X
               0017C8    +41         '0000000000000000F0F1F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0F0'X
               0017F0    +81         '00050000004D0000000000000000000000000000000000000000000000000000000000000000000000000000'X
               001818    +121        '00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'X
               001840    +161        '0000004B0000000001000000000000000000000000000000000000000000000000000000000000000000000000'X
               001868    +201        '000000000000000000000000000000000000000000000000000000000000000000000000000000000000002900'X
```

Figure 22 (Part 7 of 10). Example of a COBOL Formatted Dump

```
FORMATTED DATA DUMP FOR PROGRAM EXMPLEDUMP.QTEMP
NAME           OFFSET  ATTRIBUTES    VALUE
               001890    +241        '00000001000000000000000000000000'X
.PGTIND        0017D0  CHAR(1)       DIMENSION(32)
               0017D0    (1)         '0'
               0017D1    (2)         '1'
               0017EF    (3-32)      '0'
.PGTINVC       0017F0  BINARY(2)     5 L
.PGTLVL        0017A0  CHAR(16)      'PGT 00.0        '
.PGTMGTL       0017C0  POINTER(SPP)  NULL
.PGTMGT1       0017B0  POINTER(SPP)  SPACE OFFSET    832     '00000340'X
                                     OBJECT    PSSA
.PNP0003       000C60  CHAR(48)      '               4               F               '
               000C60  VALUE IN HEX  '400000000000000002613492D0005F44000000000000000002613492D0007C60003000000000000'X
               000C88    +41         '0000000000000000'X
.PTABLE        000C50  CHAR(16)      'PT 01.0    0    '              'D7E340F0F14BF000100001F000000000'X
.PTHSIZE       000C57  BINARY(2)     16
.PTNUM         000C59  BINARY(2)     1
.PTSEG         000C5B  CHAR(1)       '0'
.P020001       000D80  PACKED(2,0)   25
.QCRDISP       000D70  POINTER(SYP)  OBJECT    QCRADRTN
                                     CONTEXT   QSYS
.QCRXHAN       000E30  POINTER(SYP)  OBJECT    QCREXHAN
                                     CONTEXT   QSYS
.RETURNP       000960  POINTER(IP)   NULL
.RTNPTR        000690  POINTER(SPP)  SPACE OFFSET    11728     '00002DD0'X
                                     OBJECT    PASA
.RUNRTN        000970  POINTER(IP)   NULL
.SAVKKEY       0008E7  CHAR(121)     '                                                                                                '
               000941    +91         '                                '
               0008E7  VALUE IN HEX  '000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'X
               00090F    +41         3 LINES OF ZEROES SUPPRESSED
.SEPTP         000540  POINTER(SPP)  NULL
.SIZERP        0006D0  POINTER(SPP)  SPACE OFFSET    1680     '00000690'X
                                     OBJECT    PSSA
.SUBLEN        000DF0  BINARY(2)     0
.SUBNAME       000DF2  CHAR(10)      '          '                    '00000000000000000000'X
.SUBTXT        000DF0  CHAR(12)      '            '                  '000000000000000000000000'X
.SUBTXTA       000E00  POINTER(SPP)  SPACE OFFSET    3568     '00000DF0'X
                                     OBJECT    PSSA
.SUB2FST       000DC2  CHAR(10)      '          '                    '00000000000000000000'X
.SUB2LEN       000DC0  BINARY(2)     0
.SUB2SCD       000DCC  CHAR(10)      '          '                    '00000000000000000000'X
.SUB2TP        000DE0  POINTER(SPP)  SPACE OFFSET    3520     '00000DC0'X
                                     OBJECT    PSSA
.SUB2TXT       000DC0  CHAR(22)      '                      '       '0000000000000000000000000000000000000000000000'X
.TC00001       000CD4  CHAR(2)       '  '                           '0000'X
.TC00002       000CD6  CHAR(2)       '  '                           '0000'X
.TMPN001       000D80  CHAR(32)      ' ¬                             '
               000D80  VALUE IN HEX  '025F00000000000000000000000000000000000000000000000000000000000000000000'X
.T000001               NOT ADDRESSABLE
.T1            000620  CHAR(32)      '                               '
               000620  VALUE IN HEX  '00000000000000000000000000000000000000000000000000000000000000000000000000'X
.T2            000640  CHAR(32)      '                               '
               000640  VALUE IN HEX  '00000000000000000000000000000000000000000000000000000000000000000000000000'X
.T3            000660  CHAR(32)      '                               '
               000660  VALUE IN HEX  '00000000000000000000000000000000000000000000000000000000000000000000000000'X
.UCB           000B10  CHAR(32767)   '           ¢             ¢  -           ¢  -           ¢  -           ¢            '
```

Figure 22 (Part 8 of 10). Example of a COBOL Formatted Dump

```
FORMATTED DATA DUMP FOR PROGRAM EXMPLEDUMP.QTEMP
NAME           OFFSET  ATTRIBUTES    VALUE
               000B6A    +91         '                                    SALESFILE   *LIBL                                0100'
               000BC4    +181        '                                                "  ¬   00311111080062222216'
               000B10  VALUE IN HEX  '8000000000000000026C0194A0003B0800000000000000000026C0194A000860800000000000000000'X
               000B38    +41         '0026C0194A000860800000000000000000026C0194A000460800000000000000000026C0194A000576'X
               000B60    +81         '00000000000000000000000000000000000000000000000000000000000000000000000000000000'X
               000B88    +121        '0000000000000000E2C1D3C5E2C6C9D3C540FFB55CD3C9C2D34040404040FFB940404040404040404040'X
               000BB0    +161        '4040404040404040404040404000010120F0F1F0F000008D1120000000000000000000000000000000'X
               000BD8    +201        '00000000000000000000001000C00148000038000 3C80003A8000000006007FFF015F030F00F0F0F3F1'X
               000C00    +241        'F1F1F1F1F0F8F0F0F6F2F2F2F2F2F1F6'X
.UCBCLMG       000BC8  CHAR(1)       ' '                              '20'X
.UCBEDOP       000BC9  CHAR(1)       ' '                              '00'X
.UCBFILE       000B90  CHAR(10)      'SALESFILE '
.UCBFLGS       000BBE  CHAR(2)       '  '                             '0120'X
.UCBFLG1       000BBE  CHAR(1)       ' '                              '01'X
.UCBFLG2       000BBF  CHAR(1)       ' '                              '20'X
.UCBIBR@       000B20  POINTER(SPP)  SPACE OFFSET    1200     '000004B0'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.UCBINDX       000BBC  BINARY(2)     1
.UCBIOF@       000B50  POINTER(SPP)  SPACE OFFSET    454     '000001C6'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.UCBLAST       000BB2  CHAR(10)      '          '
.UCBLBID       000B9A  BINARY(2)     -75
.UCBLIB        000B9C  CHAR(10)      '*LIBL     '
.UCBLIBS       000B9A  CHAR(12)      '  *LIBL     '                   'FFB55CD3C9C2D34040404040'X
.UCBMBID       000BA6  BINARY(2)     -71
.UCBMBRS       000BA6  CHAR(12)      '            '                   'FFB940404040404040404040'X
.UCBMLIB       000BA8  CHAR(10)      '          '
.UCBNXT@       000B60  POINTER(SPP)  NULL
.UCBOBR@       000B30  POINTER(SPP)  SPACE OFFSET    1200     '000004B0'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.UCBODP@       000B10  POINTER(SPP)  SPACE OFFSET    0     '00000000'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.UCBOPF@       000B40  POINTER(SPP)  SPACE OFFSET    176     '000000B0'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.UCBPARM       000BE0  BINARY(2)     1
.UCBRLEN       000BE2  BINARY(2)     12
.UCBRLVR       000BC0  CHAR(4)       '0100'
.UCBSEP@       000B70  POINTER(SPP)  NULL
.UFCBPTR       000860  POINTER(SPP)  SPACE OFFSET    2832     '00000B10'X
                                     OBJECT    PSSA
.UFLGSAV       000DA0  CHAR(2)       '  '                             '0000'X
.USERTN        000720  POINTER(IP)   NULL
.USEWRK@       000E40  POINTER(SPP)  NULL
.U01CLMG       000BC8  CHAR(1)       ' '                              '20'X
.U01FLGS       000BBE  CHAR(2)       '  '                             '0120'X
.U01IBF@       000B20  POINTER(SPP)  SPACE OFFSET    1200     '000004B0'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.U01OBF@       000B30  POINTER(SPP)  SPACE OFFSET    1200     '000004B0'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.U01UFCB       000B10  POINTER(SPP)  SPACE OFFSET    0     '00000000'X
                                     OBJECT    SALESFILE COB38EX   SALESFILE
.V0054FA       000652  CHAR(1)       '5'
.WCBCNLS       0007D0  CHAR(1)       '0'
.WCBJDAT       0007D1  CHAR(7)       '0880826' M
.WCBLURC       0007C0  BINARY(2)     2
.WCBPINF       0007C2  BINARY(2)     0
```

Figure 22 (Part 9 of 10). Example of a COBOL Formatted Dump

```
FORMATTED DATA DUMP FOR PROGRAM EXMPLEDUMP.QTEMP
NAME           OFFSET  ATTRIBUTES    VALUE
.WCBSWTC       0007D8  CHAR(8)       '00000000'
.WCBUDTA       0007C0  CHAR(32767)   '               0088021700000000CR                                 _                                    '
               00081A    +91         2 LINES OF BLANKS SUPPRESSED
               0007C0  VALUE IN HEX  '0002000000000000000000000000000F0F0F8F8F0F2F1F7F0F0F0F0F0F0F0F0C3D9000000000000'X
               0007E8    +41         '00000000000000000000000000000000026A06D230019FF0000000000000000000000000000000'X
               000810    +81         5 LINES OF ZEROES SUPPRESSED
.WCBURC        0007CE  CHAR(2)       '  '                            '0000'X
.WCBU0         0007D8  CHAR(1)       '0'
.WCBU1         0007D9  CHAR(1)       '0'
.WCBU2         0007DA  CHAR(1)       '0'
.WCBU3         0007DB  CHAR(1)       '0'
.WCBU4         0007DC  CHAR(1)       '0'
.WCBU5         0007DD  CHAR(1)       '0'
.WCBU6         0007DE  CHAR(1)       '0'
.WCBU7         0007DF  CHAR(1)       '0'
END-FLAG       000C4C  CHAR(1)       ' '  N
END-OF-INPUT   000C4D  CHAR(1)       'Y'
FILE-1         000650  CHAR(12)      'H2500000000 '
FILLER         000C2E  CHAR(3)       '   '
FILLER         000C3D  CHAR(3)       '   '
FILLER         00065B  CHAR(1)       ' '
FILLER         000C1A  CHAR(8)       'TOTALS: '
R-AREA-CODE    000651  ZONED(2,0)    25
R-NORTH-EAST   000BF9  PACKED(2,0)   30
R-NORTH-EAST   000BF7  PACKED(2,0)   15
R-SALES-CAT-1  000653  PACKED(7,2)
                     **INVALID DATA  'F0F0F0F0'X  0
R-SALES-CAT-2  000657  PACKED(7,2)
                     **INVALID DATA  'F0F0F0F0'X
R-TYPE         000650  CHAR(1)       'H'
RECORD-1       000650  CHAR(12)      'H2500000000 '
W-CAT-1        000BFC  ZONED(10,2)   311111.08
W-CAT-2        000C06  ZONED(10,2)   622222.16
W-EDIT-TOTAL   000C40  CHAR(12)      '            '
W-EDIT-VALUES  000C1A  CHAR(50)      'TOTALS:                                          '
W-EDIT-1       000C22  CHAR(12)      '            '
W-EDIT-2       000C31  CHAR(12)      '            '
W-SALES-VALUES 000BFC  CHAR(30)      '003111110800622222160093333324'
W-TOTAL        000C10  ZONED(10,2)   933333.24
STATIC STORAGE FOR PROGRAM EXMPLEDUMP.QTEMP      BEGINS AT OFFSET 000340 IN THE PROGRAM STATIC STORAGE AREA (PSSA)
AUTOMATIC STORAGE FOR PROGRAM EXMPLEDUMP.QTEMP      BEGINS AT OFFSET 002E10 IN THE PROGRAM AUTOMATIC STORAGE AREA (PASA)
```

Figure 22 (Part 10 of 10). Example of a COBOL Formatted Dump

Identifying COBOL Problems

When a COBOL problem occurs, you can use the following series of questions to pinpoint its possible cause:

Have changes been made to the user program since the last time it compiled or ran successfully?

NO YES

> Read on, but consider what has been changed. For example, have operating procedures changed, are new device files being used, or have program changes been applied recently? A good starting point for problem determination can be a changed item.

Are you using the current release of the COBOL compiler? The release number is printed on the first line of the source listing.

YES NO

> Install the current release of the compiler and the program changes that apply to the release, and recompile the program. Refer to the *Software Installation*, for an explanation of installing the compiler.

Did the COBOL compiler have an exception?

NO YES

An exception is an error indicating an abnormal compiler termination. The exception is displayed at the work station or printed in the job log for the job that requested the compile. Exception data is printed and can be used to investigate the problem. For example, the name of the compiler phase being run and the number of the COBOL source statement being processed at the time of the exception are printed. You can refer to the source statement in the user program and try to modify the code to circumvent the problem. Retry the compilation and specify the DUMP parameter on the Create COBOL Program command. Whether or not you successfully circumvent the problem, you should report it to your service representative.

Did the COBOL compiler loop or wait while compiling a user program?

NO YES

A loop or a wait is a seemingly never-ending compilation for which neither output nor error messages are produced.

For either condition you can (1) enter service mode for the job, (2) request a dump of the job, and (3) cancel the job. Refer to the *Operator's Quick Reference*, for descriptions of entering service mode, requesting a dump, and canceling a job.

Use the program stack layout at the beginning of the dump to investigate the problem.

Did the COBOL compiler produce incorrect output?

NO YES

One of the following conditions can indicate incorrect compiler output:

- An IRP syntax error, which causes the compiler to end abnormally and an error message to be issued to the compiler requester.
- An unexpected result produced by a user program while it is run.

IRP Syntax Error: If your program caused an IRP syntax error, the IRP will be listed and you can trace the incorrect IRP statement back to the COBOL source statement(s) that produced it. Try to modify the COBOL source program to circumvent the error and then recompile the program.

Unexpected Results: If a user program produces unexpected results, you can use OS/400 debugging functions such as traces and breakpoints, or you can use the COBOL debugging feature USE FOR DEBUGGING statement to isolate problems within a user program. Refer to "DEBUGGING FEATURES" on page 517 for a description of COBOL debugging features and to "Debugging Your Program" on page 60 for an overview of OS/400 debugging functions.

If after investigating the problem you suspect a compiler error, try to modify the COBOL source program to circumvent the error and then recompile the program.

Even if you circumvent the problem, you should also report it to your service representative.

Did the COBOL user program have an exception/error?

NO YES

Two types of exception/error can occur: program and file. Examples of program exception/errors are division by zero, use of an invalid index, and use of uninitialized data items in an arithmetic operation. Examples of file exception/errors are undefined record types and device errors.

You can begin investigating the problem at the source statement indicated by the message. Use OS/400 debugging functions or COBOL debugging features to pinpoint the problem. "DEBUGGING FEATURES" on page 517 describes COBOL debugging features, and "Debugging Your Program" on page 60 provides an overview of OS/400 debugging functions.

Did the COBOL user program loop, or wait while it was being run, or produce incorrect output?

NO YES

A loop is a sequence of instructions that is processed repeatedly, and a wait is a situation in which neither output nor error messages are produced.

For either condition you can (1) enter service mode for the job, (2) request a dump of the job, and (3) cancel the job. Refer to the *Operator's Quick Reference*, for descriptions of entering service mode, requesting a dump, and canceling a job.

Use the program stack at the beginning of the dump to investigate the problem. Also check for program logic errors; if the program appears to be coded correctly, you can use OS/400 debugging functions or COBOL debugging features to pinpoint the problem. For example, you can trace the processing of certain statements and display the contents of fields at stopping points in the program. Refer to Chapter 11, "Using the Additional COBOL Functions" for a description of COBOL debugging features and to "Debugging Your Program" for an overview of OS/400 debugging functions.

When you isolate the problem to one or more source statements, try to modify the COBOL source program to circumvent the error and then recompile the program.

Refer to the section "Calling for Help" which follows.

Calling for Help

If you require additional assistance, do the following:

1. Cancel the failing job and print the job log, if you have not already done so. Do this by signing off your work station and choosing *LIST for the LOG parameter. For example:

   ```
   SIGNOFF LOG(*LIST)
   ```

   Call your system operator to verify that the job log was printed.

2. Examine the job log, and any other available information on your job, to determine why your problem occurred. If you still require additional assistance, go to the next step.

3. Before calling for service, have your system operator refer to the *Operator's Quick Reference*, for further problem analysis guidance.

Chapter 5. Interactive Processing Considerations and Example Programs

IBM Extension

This chapter describes the System/38-Compatible COBOL language extensions that support work stations and program-to-program communication.

You might need to refer to other AS/400 manuals for information about a particular topic in this chapter. They are listed below:

- *Communications: Advanced Peer-to-Peer Networking Guide*, which is intended for the programmer responsible for defining or using OS/400 Advanced Peer-to-Peer Networking* (APPN*). For writing application programs that use OS/400 advanced program-to-program communications (APPC), see *Communications: Advanced Program-to-Program Communications Programmer's Guide*.
- *DDS Reference*, which describes the data description specifications (DDS) that are used for describing files.
- *Data Management Guide*, which contains information about overriding and copying files, describing display, printer, tape, and diskette files to the system, as well as spooling and output queues.

In addition, you might need to refer to the following System/38 publications for information about a particular topic in this chapter which would pertain to the AS/400 System/38 environment. They are listed below:

- *IBM System/38 Control Program Facility Programmer's Guide*, SC21-7730, which explains how to use CPF commands and data description specifications.
- *IBM System/38 Control Language Reference Manual*, SC21-7731, which describes commands and parameters that are used for various CPF functions.
- *IBM System/38 Control Program Facility Reference Manual*, SC21-7806, which describes the data description specifications that are used for describing files.
- *IBM System/38 Data Communications Programmer's Guide*, SC21-7825, which describes commands, parameters, and data description specification keywords that are used for program-to-program and system-to-device communication functions.

The TRANSACTION file organization allows a COBOL program to interactively communicate with:

- One or more work station users
- One or more programs on a remote system
- One or more devices on a remote system
- Any combination of the above.

On the AS/400 system you communicate with a program or device on a remote system by using LU1, BSC, or PEER (APPC) devices.

When required, references are made to prior discussions in earlier chapters of this book, in order to avoid repetition. The language extensions in this chapter are pre-

sented in the following sequence: Environment Division, Data Division, and Procedure Division.

Externally Described Transaction File

A COBOL TRANSACTION file normally uses an externally described display file, BSC file, communications file, or mixed file that contains file information and a description of the fields in the records. The records in this file are described to the COBOL program by the Format 2 of the COPY statement (see the format diagram in Figure 5 on page 32).

In addition to the field descriptions (such as field names and attributes), the Data Description Specifications (DDS) for a display device file:

- Specify the line number and position number entries for each field and constant to format the placement of the record on the screen.
- Specify attention functions such as underlining and highlighting fields, reverse image, or a blinking cursor.
- Specify validity checking for data entered at the display work station. Validity checking functions include:
  - Detecting fields where data is required
  - Detecting mandatory fill fields
  - Detecting incorrect data types
  - Detecting data for a specific range
  - Checking data for a valid entry
  - Performing modulus 10 or 11 check digit verification.
- Control screen management functions such as when fields are to be erased, overlaid, or retained when new data is displayed.
- Associate indicators 01 through 99 with function keys designated as type CA or CF. If a function key is designated as CF, both the modified data record and the response indicator are returned to the program. If a function key is designated as CA, the response indicator is returned to the program, but the data record usually contains default values for input-only fields and the values written to the format for hidden output/input fields.
- Assign an edit code (EDTCDE keyword) or edit word (EDTWRD keyword) to a field to specify how the field's values are to be displayed.
- Specify subfiles.

A display device record format contains three types of fields:

- *Input Fields:* Input fields pass from the device to the program when the program reads a record. Input fields can be initialized with a default value; if the default value is not changed, the default value passes to the program. Uninitialized input fields are displayed as blanks into which the work station user can enter data.
- *Output Fields:* Output fields pass from the program to the device when the program writes a record to a display. The program or the record format in the device file can provide output fields.
- *Output/Input (both) Fields:* An output/input field is an output field that can be changed and, therefore, become an input field. Output/input fields pass from

the program when the program writes a record to a display and pass to the program when the program reads a record from the display. Output/input fields are used when the user is to change or update the data that is written to the display from the program.

Figure 23 shows an example of the DDS for a display device file.

IBM International Business Machines — **DATA DESCRIPTION SPECIFICATIONS** — GX21-7754-1 UM/060* Printed in U.S.A. *Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic / Key	Description	Page of
Programmer	Date				

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Not (N) / Indicator (Conditioning)	Name Type (b/R/K/S/O)	Name	Reference (R)	Length	Data Type	Decimal Positions	Usage (b/O/I/B/H/M)	Line	Pos	Functions
	A	* CUSTOMER MASTER INQUIRY FILE -- CUSMINQ											
	A	*											
	A												REF(CUSMSTP) **1**
	A			R	CUSPMT								TEXT('CUSTOMER PROMPT')
	A												CA01(15 'END OF PROGRAM') **2**
	A										1	3	'CUSTOMER MASTER INQUIRY'
	A										3	3	'CUSTOMER NUMBER'
	A				CUST	R				I	3	20	
	A		99										ERRMSG('CUSTOMER NUMBER NOT FOUND+ **3**
	A												PRESS RESET, THEN ENTER VALID NUMBE+
	A												R' 99)
	A										5	3	'USE F1 TO END PROGRAM, USE ENTER +
	A												KEY TO RETURN TO PROMPT SCREEN'
	A			R	CUSFLDS								TEXT('CUSTOMER DISPLAY')
	A												CA01(15 'END OF PROGRAM')
	A												OVERLAY **4**
	A										8	3	'NAME'
	A				NAME	R					8	11	
	A										9	3	'ADDRESS'
	A				ADDR	R					9	11	
	A										10	3	'CITY' **5**
	A				CITY	R					10	11	
	A									**6**	11	3	'STATE'
	A				STATE	R					11	11	
	A										11	21	'ZIP CODE'
	A				ZIP	R					11	31	
	A										12	3	'A/R BALANCE'
	A				ARBAL	R					12	17	EDTCDE(J)
	A												
	A												

This display device file contains two record formats: CUSPMT and CUSFLDS.

1 The attributes for the fields in this file are defined in the CUSMSTP field reference file.

2 Function key 1 (CA01) is associated with indicator 15, with which the user ends the program.

3 The ERRMSG keyword identifies the error message that is displayed if indicator 99 is set on in the program that uses this record format.

4 The OVERLAY keyword is used for the record format CUSFLDS so that the CUSPMT record on the display will not be erased when the CUSFLDS record is written to the display.

5 The constants such as 'Name', 'Address', and 'City' describe the fields that are written out by the program.

6 The line and position entries identify where the fields or constants are written on the display.

Figure 23. Example of the Data Description Specifications for a Display Device File

Processing an Externally Described TRANSACTION File

When an externally described TRANSACTION file is processed, Operating System/400 transforms data from the program to the format specified for the file and displays the data. When data passes to the program, the data is transformed to the format used by the program.

OS/400 provides device control information for processing input/output operations for the device. When an input record is requested from the device, OS/400 issues the request, then removes device control information from the data before passing the data to the program. In addition, OS/400 can pass indicators to the program indicating which fields, or if any fields, in the record have changed.

When the program requests an output operation, it passes the output record to OS/400. OS/400 provides the necessary device control information to display the record. OS/400 also adds any constant information specified for the record format when the record is displayed.

When a record passes to a program, the fields are arranged in the order in which they are specified in the DDS. The order in which the fields are displayed is based on the display positions (line numbers and positions) assigned to the fields in the DDS. Therefore, the order in which the fields are specified in the DDS and the order in which they appear on the screen need not be the same.

Indicators

Indicators are Boolean data items that can have the values B"0" or B"1".

When you define a record format for a file using DDS, you decide the options that are to be controlled by indicators, and the indicators that are to reflect particular responses.

Option indicators provide options such as spacing, underlining, and allowing or requesting data transfer from a program to a printer or display device. Response indicators provide response information to a program from a device, such as which function keys are pressed by a work station user, and whether data has been entered.

Indicators can also be used with FORMATFILE files.

Indicators can be passed with data records in a record area, or outside the record area in a separate indicator area.

Indicators in the Record Area

If the keyword INDARA was not used in the DDS for a file, indicators are created in the record area. When indicators are defined in a record format for a file, they are read, rewritten, and written with the data in the record area.

The number and order of indicators defined in the DDS for a record format for a file determines the number and order in which the data description entries for the indicators in the record format must be coded in the program.

If a COPY statement, Format 2, is used to copy indicators into a source program, the indicators are defined in the order in which they are specified in the DDS for the file.

Indicators in a Separate Indicator Area

If the file level keyword INDARA was specified in the DDS, then for any record format in the file, all indicators are passed to and from the program in a separate indicator area, not in the record area.

The file control entry for a file that had INDARA specified in its DDS must have the separate indicator area attribute, SI, as part of the assignment-name in the ASSIGN clause.

The advantages of using a separate indicator area are as follows:

- The number and order of indicators used in an I-O statement for any record format in a file need not match the number and order of indicators specified in the DDS for that record format.
- The program associates the indicator number in a data description entry with the appropriate indicator.

ASSIGN Clause with Separate Indicator Area Attribute

Assignment-name of the ASSIGN clause of a file control entry has the following general format:

```
device–file name–SI
```

Device can be either WORKSTATION or FORMATFILE.

file name must refer to a file that has the file level keyword INDARA specified in its DDS.

An example of an assignment-name is:

```
WORKSTATION–INVSCRNS–SI
```

Data Description Entry–Boolean Data

When you use indicators in a COBOL program, you must describe them as Boolean data items using the data description entry for Boolean data.

Format 4 - Boolean Data

```
level-number { data-name-1 }
             { FILLER      }

      [ REDEFINES data-name-2 ]

      [ { PICTURE } IS 1 ]
      [ { PIC    }       ]

      [ [ USAGE IS ] DISPLAY ]

      [ OCCURS { integer-1 TO integer-2 TIMES DEPENDING ON data-name-3 }
               { integer-2 TIMES                                       }

               [ INDEXED BY index-name-1 [ , index-name-2 ] . . . ] ]

      [ { INDICATOR  }            ]
      [ { INDICATORS } integer-3  ]
      [ { INDIC      }            ]

     **********************************
     * [                            ] *
     * |                            | *
     * | { SYNCHRONIZED } [ LEFT  ] | *
     * | { SYNC          } [ RIGHT ] | *
     * |                            | *
     * [                            ] *
     *                                *
     * [                            ] *
     * | { JUSTIFIED } RIGHT        | *
     * | { JUST      }              | *
     * [                            ] *
     **********************************

      [ VALUE IS Boolean-literal ] .
```

Special Considerations

The special considerations for the clauses used with the Boolean data follow. All other rules for clauses are the same as those for other data as described under “Data Description Entry” in Chapter 9, “Data Division”

PICTURE Clause: An elementary Boolean data-name is defined by a PICTURE containing a single 1.

USAGE Clause: USAGE must be defined implicitly or explicitly as DISPLAY.

OCCURS Clause: When the OCCURS clause and the INDICATOR clause are both specified at an elementary level, a table of Boolean data items is defined with each element in the table corresponding to an external indicator. The first element in the table corresponds to the indicator number specified in the INDICATOR clause; the second element corresponds to the indicator that sequentially follows the indicator specified by the INDICATOR clause.

For example, if the following is coded:

```
07     SWITCHES      PIC 1
                     OCCURS 10 TIMES
                     INDICATOR 16
```

then SWITCHES (1) corresponds to indicator 16, SWITCHES (2) corresponds to indicator 17,..., and SWITCHES (10) corresponds to indicator 25.

INDICATOR Clause: If indicator fields are in a separate indicator area, the INDICATOR clause associates an indicator defined in DDS with a Boolean data item. If indicator fields are in the record area, the INDICATOR clause is syntax-checked, but is treated as documentation.

Integer-3 must be a value of 1 through 99.

The INDICATOR clause must be specified at an elementary level only.

VALUE Clause: The VALUE clause specifies the initial content of a Boolean data item. The allowable values for Boolean literals are B"0", B"1", and ZERO.

INDICATORS Phrase

When the INDICATORS phrase is used in READ, REWRITE, and WRITE statements (see Figure 26 on page 100) it specifies which indicators are to be read, rewritten, and written.

The identifier specified in the INDICATORS phrase can be either of the following:

- An elementary Boolean data item
- A group item with elementary Boolean data items subordinate to it.

Indicators in the Record Area

If INDARA was not specified in the DDS for the file, the size of the identifier in the INDICATORS phrase of an I-O statement (see Figure 26 on page 100) should be equal to the number of option or response indicators defined in the DDS for that format.

- In a READ statement, the identifier size should be equal to the number of response indicators.
- In a REWRITE or WRITE statement, the identifier size should be equal to the number of option indicators.

The contents of the identifier are not checked, but are copied to or from the beginning of the record, on a byte by byte basis; indicator numbers are ignored.

If the INDICATORS phrase is omitted, the data in the indicator fields in the record are still passed in the record area. The INDICATORS phrase is only used to copy indicators into the record area before a WRITE or REWRITE statement, or out of the record area after a READ statement.

Indicators in a Separate Indicator Area

If INDARA was specified in the DDS for the file, the use of the indicators referenced in the INDICATORS phrase is based on indicator number.

- In a READ statement, only the response indicator numbers referenced by the INDICATORS phrase are updated. Indicators specified in the DDS for the format but not referenced by the INDICATORS phrase are ignored. Indicators referenced by the INDICATORS phrase but not specified in the DDS are not modified.
- In a WRITE or REWRITE statement, only the option indicators referenced by the INDICATORS phrase are used. Indicators specified in the DDS for the format but not referenced by the INDICATORS phrase are assumed to be off. Indicators referenced by the INDICATORS phrase but not used in the DDS for the format are ignored.

If the INDICATORS phrase is not specified, the following occurs:

- In the READ statement, indicators are not updated.
- In a WRITE or REWRITE statement, indicators are treated as through they are set off.

Indicators Example Programs

This section contains COBOL program examples that illustrate the use of indicators in either a record area or a separate indicator area.

All the programs do the following:

1. Determine the current date.
2. If it is the first day of the month, turn on an option indicator that causes an output field to appear and blink.
3. Allow the work station user to press F keys to terminate the program, or turn on response indicators and call programs to write daily or monthly reports.

Figure 25 on page 98 shows a program that uses indicators in the record area but does not use the INDICATORS phrase in any I-O statement. The associated DDS for the file is shown in Figure 24 on page 97.

Figure 26 on page 100 shows a program that uses indicators in the record area and the INDICATORS phrase in the I-O statements. The associated DDS for the file is the same as that shown for the program in Figure 24 on page 97.

Figure 28 on page 103 shows a program that uses indicators in a separate indicator area, defined in WORKING-STORAGE by using the Format 2 COPY statement. The associated DDS for the file is shown in Figure 27 on page 102.

Figure 29 on page 105 shows a program that uses indicators in a separate indicator area, defined in a table in WORKING-STORAGE. The associated DDS for the file is the same as that shown for the program in Figure 27 on page 102.

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7754-1 UM/080*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Not (N)	Condition Name Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (b/R/K/S/C)	Reserved	Name	Reference (R)	Length	Data Type (b A/P/S/B A/S/X/Y/N/I/U/W)	Decimal Positions	Usage (b/O/I/B/H/M)	Location Line	Location Pos	Functions
	A	*									DISPLAY FILE DDS FOR INDICATOR EXAMPLES								
	A	*																1	
	A					2	R	FORMAT1										3	CF01(99 'END OF PROGRAM')
	A																		CF05(51 'DAILY REPORT')
	A																		CF09(52 'MONTHLY REPORT')
	A	*																	
	A																10	10	'DEPARTMENT NUMBER: '
	A										DEPTNO		5	4	I		10	32	
	A																20	26	'PRODUCE MONTHLY REPORTS'
	A			5	01														DSPATR(BL)
	A	*																	
	A													6			24	01	'F5 = DAILY REPORT'
	A																24	26	'F9 = MONTHLY REPORT'
	A																24	53	'F1 = TERMINATE'
	A																		
	A																		
	A																		

1 The INDARA keyword is not used; indicators are stored in the record area with the data fields.

2 One record format, FORMAT1, is specified.

3 Three indicators are associated with three F keys. Indicator 99 will be set on when F key 1 is pressed, and so on.

4 One field is defined for input.

5 Indicator 01 is defined to cause the associated constant field to blink if the indicator is on.

6 The F key definitions are documented on the work station screen.

Figure 24. Example of a Program Using Indicators in the Record Area without Using the INDICATORS Phrase in the I-O Statement–Data Description Specifications

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000110   PROGRAM-ID.     EXMPLE710.
       000120*    EXAMPLE PROGRAM WITH INDICATORS IN RECORD AREA.
    3  000130   AUTHOR.       PROGRAMMER NAME.
    4  000140   INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
    5  000150   DATE-WRITTEN. 08/30/88.
    6  000160   DATE-COMPILED.  08/30/88 09:30:00   .
    7  000170 ENVIRONMENT DIVISION.
    8  000180   CONFIGURATION SECTION.
    9  000190   SOURCE-COMPUTER. IBM-S38.
   10  000200   OBJECT-COMPUTER. IBM-S38.
   11  000210   INPUT-OUTPUT SECTION.
   12  000220   FILE-CONTROL.
   13  000230     SELECT DISPFILE
   14  000240       ASSIGN TO WORKSTATION-DISPFILEX _
   15  000250       ORGANIZATION IS TRANSACTION
   16  000260       ACCESS       IS SEQUENTIAL.
   17  000270 DATA DIVISION.
   18  000280   FILE SECTION.
   19  000290   FD  DISPFILE
   20  000300     LABEL RECORDS ARE OMITTED
   21  000310     DATA RECORD IS DISP-REC.
   22  000320   01  DISP-REC.
   23  000330       COPY DDS-ALL-FORMATS OF DISPFILEX. __
   24 +000001       05  DISPFILEX-RECORD PIC X(8).                                    <-ALL-FMTS
      +000002*  INPUT FORMAT:FORMAT1    FROM FILE DISPFILEX  OF LIBRARY EXMPLIB      <-ALL-FMTS
      +000003*                                                                       <-ALL-FMTS
   25 +000004       05  FORMAT1-I     REDEFINES DISPFILEX-RECORD.                   <-ALL-FMTS
   26 +000005           06 FORMAT1-I-INDIC.                                         <-ALL-FMTS
   27 +000006                07 IN99         PIC 1  INDIC 99. ___                  <-ALL-FMTS
      +000007*                           END OF PROGRAM                           <-ALL-FMTS
   28 +000008                07 IN51         PIC 1  INDIC 51.                      <-ALL-FMTS
      +000009*                           DAILY REPORT                             <-ALL-FMTS
   29 +000010                07 IN52         PIC 1  INDIC 52.                      <-ALL-FMTS
      +000011*                           MONTHLY REPORT                           <-ALL-FMTS
   30 +000012           06 DEPTNO            PIC X(5).                             <-ALL-FMTS
      +000013* OUTPUT FORMAT:FORMAT1    FROM FILE DISPFILEX  OF LIBRARY EXMPLIB      <-ALL-FMTS
      +000014*                                                                       <-ALL-FMTS
   31 +000015       05  FORMAT1-O     REDEFINES DISPFILEX-RECORD.                   <-ALL-FMTS
   32 +000016           06 FORMAT1-O-INDIC.                                         <-ALL-FMTS
   33 +000017                07 IN01         PIC 1  INDIC 01.                      <-ALL-FMTS
   34  000340
   35  000350 WORKING-STORAGE SECTION.
   36  000360 01  CURRENT-DATE.
   37  000370    05  CURR-YEAR                 PIC 9(2).
   38  000380    05  CURR-MONTH                PIC 9(2).
   39  000390    05  CURR-DAY                  PIC 9(2).
   40  000400 01  INDIC-AREA. ____
   41  000410    05  IN01                      PIC 1.
   42  000420 _____     88  NEW-MONTH             VALUE B"1".
   43  000430    05  IN51                      PIC 1.
   44  000440       88  WANT-DAILY             VALUE B"1".
   45  000450    05  IN52                      PIC 1.
   46  000460       88  WANT-MONTHLY           VALUE B"1".
   47  000470    05  IN99                      PIC 1.
```

Figure 25 (Part 1 of 2). Example of a Program Using Indicators in the Record Area without Using the INDICATORS Phrase in the I-O Statement

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   48  000480       88  NOT-END-OF-JOB          VALUE B"0".
   49  000490       88  END-OF-JOB              VALUE B"1".
   50  000500 PROCEDURE DIVISION.
       000510 SAMPLE3-MAIN.
   51  000520     OPEN I-O DISPFILE.
   52  000530     ACCEPT CURRENT-DATE FROM DATE.
   53  000540     SET NOT-END-OF-JOB TO TRUE.
   54  000550     PERFORM DISPLAY-SCREEN THRU READ-AND-PROCESS-SCREEN
       000560             UNTIL END-OF-JOB.
   55  000570     CLOSE DISPFILE.
   56  000580     STOP RUN.
       000590 DISPLAY-SCREEN.
   57  000600  6    MOVE ZEROS TO INDIC-AREA.
   58  000610     IF CURR-DAY = 01 THEN
   59  000620  7       SET NEW-MONTH TO TRUE.
   60  000630  8    MOVE CORR INDIC-AREA TO FORMAT1-O-INDIC.
   61  000640 _____    WRITE DISP-REC FORMAT IS "FORMAT1".  10
       000650 READ-AND-PROCESS-SCREEN.
   62  000660     MOVE ZEROS TO INDIC-AREA.
   63  000670 C/xx)  READ DISPFILE FORMAT IS "FORMAT1".
   64  000680 Draft  MOVE CORR FORMAT1-I-INDIC TO INDIC-AREA.
   65  000690     IF WANT-DAILY THEN
   66  000700  13     CALL "DAILY" USING DEPTNO
       000710     ELSE
   67  000720        IF WANT-MONTHLY THEN
   68  000730           CALL "MONTHLY" USING DEPTNO.
                        * * * * *  E N D  O F  S O U R C E  * * * * *
```

1 The separate indicator area attribute, `SI`, is not coded in the `ASSIGN` clause.

2 The Format 2 `COPY` statement, defines data fields and indicators in the record area.

3 Because the file does not have a separate indicator area, response and option indicators are defined in the order in which they are used in the DDS, and the indicator numbers are treated as documentation.

4 All indicators used by the program are defined with meaningful names in data description entries in `WORKING-STORAGE`. Indicator numbers are omitted here because they have no effect.

5 For each indicator, a meaningful level 88 condition-name is associated with a value for that indicator.

6 Initialize group level to zeros.

7 `IN01` in `WORKING-STORAGE` is set on if it is the first day of the month.

8 Indicators appropriate to output of `FORMAT1` are copied to the record area.

9 `FORMAT1` is written to the work station screen with both data and indicator values in the record area.

10 The `INDICATORS` phrase is not necessary because there is no separate indicator area and indicator values have been set in the record area through the previous `MOVE CORRESPONDING` statement.

11 `FORMAT1`, including both data and indicators, is read from the screen.

12 The response indicators for `FORMAT1` are copied from the record area to the data description entries in `WORKING-STORAGE`.

13 If `F` key 5 has been pressed, a program call is processed.

Figure 25 (Part 2 of 2). Example of a Program Using Indicators in the Record Area without Using the INDICATORS Phrase in the I-O Statement

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000110   PROGRAM-ID.     EXMPLE713.
       000120*    EXAMPLE PROGRAM - FILE WITH INDICATORS IN RECORD AREA
    3  000130   AUTHOR.       PROGRAMMER NAME.
    4  000140   INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
    5  000150   DATE-WRITTEN. 08/30/88.
    6  000160   DATE-COMPILED.  08/30/88 09:31:05   .
    7  000170 ENVIRONMENT DIVISION.
    8  000180   CONFIGURATION SECTION.
    9  000190   SOURCE-COMPUTER. IBM-S38.
   10  000200   OBJECT-COMPUTER. IBM-S38.
   11  000210   INPUT-OUTPUT SECTION.
   12  000220   FILE-CONTROL.
   13  000230     SELECT DISPFILE
   14  000240       ASSIGN TO WORKSTATION-DISPFILEX _
   15  000250       ORGANIZATION IS TRANSACTION
   16  000260       ACCESS       IS SEQUENTIAL.
   17  000270 DATA DIVISION.
   18  000280   FILE SECTION.
   19  000290   FD  DISPFILE
   20  000300     LABEL RECORDS ARE OMITTED
   21  000310     DATA RECORD IS DISP-REC.
   22  000320   01  DISP-REC.
   23  000330       COPY DDS-ALL-FORMATS OF DISPFILEX. __
   24 +000001       05  DISPFILEX-RECORD PIC X(8).                                    <-ALL-FMTS
      +000002*  INPUT FORMAT:FORMAT1    FROM FILE DISPFILEX  OF LIBRARY EXMPLIB      <-ALL-FMTS
      +000003*                                                                       <-ALL-FMTS
   25 +000004       05  FORMAT1-I     REDEFINES DISPFILEX-RECORD.                   <-ALL-FMTS
   26 +000005           06 FORMAT1-I-INDIC.                                         <-ALL-FMTS
   27 +000006                07 IN99         PIC 1  INDIC 99. ___                  <-ALL-FMTS
      +000007*                           END OF PROGRAM                           <-ALL-FMTS
   28 +000008                07 IN51         PIC 1  INDIC 51.                      <-ALL-FMTS
      +000009*                           DAILY REPORT                             <-ALL-FMTS
   29 +000010                07 IN52         PIC 1  INDIC 52.                      <-ALL-FMTS
      +000011*                           MONTHLY REPORT                           <-ALL-FMTS
   30 +000012           06 DEPTNO            PIC X(5).                             <-ALL-FMTS
      +000013* OUTPUT FORMAT:FORMAT1    FROM FILE DISPFILEX  OF LIBRARY EXMPLIB      <-ALL-FMTS
      +000014*                                                                       <-ALL-FMTS
   31 +000015       05  FORMAT1-O     REDEFINES DISPFILEX-RECORD.                   <-ALL-FMTS
   32 +000016           06 FORMAT1-O-INDIC.                                         <-ALL-FMTS
   33 +000017                07 IN01         PIC 1  INDIC 01.                      <-ALL-FMTS
   34  000340
   35  000350 WORKING-STORAGE SECTION.
   36  000360 01  CURRENT-DATE.
   37  000370     05  CURR-YEAR                PIC 9(2).
   38  000380     05  CURR-MONTH               PIC 9(2).
   39  000390     05  CURR-DAY                 PIC 9(2).
   40  000400
   41  000410 77  IND-OFF                      PIC 1      VALUE B"0".
   42  000420 77  IND-ON                       PIC 1      VALUE B"1".
   43  000430
   44  000440 01  RESPONSE-INDICS.
   45  000450     05  END-OF-PROGRAM           PIC 1. ____
   46  000460     05  DAILY-REPORT             PIC 1.
   47  000470     05  MONTHLY-REPORT           PIC 1.
```

Figure 26 (Part 1 of 2). Example of a Program Using Indicators in the Record Area and the INDICATORS phrase in the I-O Statements

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   48  000480 01  OPTION-INDICS.
   49  000490     05  NEW-MONTH                PIC 1.
   50  000500
   51  000510 PROCEDURE DIVISION.
       000520 SAMPLE3-MAIN.
   52  000530     OPEN I-O DISPFILE.
   53  000540     ACCEPT CURRENT-DATE FROM DATE.
   54  000550     MOVE IND-OFF TO END-OF-PROGRAM.
   55  000560     PERFORM DISPLAY-SCREEN THRU READ-AND-PROCESS-SCREEN
       000570             UNTIL END-OF-PROGRAM = IND-ON.
   56  000580     CLOSE DISPFILE.
   57  000590     STOP RUN.
       000600
       000610 DISPLAY-SCREEN.
   58  000620     MOVE ZEROS TO OPTION-INDICS.
   59  000630 _____   IF CURR-DAY = 01 THEN
   60  000640        MOVE IND-ON TO NEW-MONTH.
   61  000650  6   WRITE DISP-REC FORMAT IS "FORMAT1"
       000660                    INDICATORS ARE OPTION-INDICS.
       000670
       000680 READ-AND-PROCESS-SCREEN.
   62  000690     MOVE ZEROS TO RESPONSE-INDICS.
   63  000700  7   READ DISPFILE FORMAT IS "FORMAT1"
       000710                  INDICATORS ARE RESPONSE-INDICS.  8
   64  000720     IF DAILY-REPORT = IND-ON THEN
   65  000730 _________       CALL "DAILY" USING DEPTNO
       000740     ELSE
   66  000750        IF MONTHLY-REPORT = IND-ON THEN
   67  000760           CALL "MONTHLY" USING DEPTNO.
                       * * * * *  E N D  O F  S O U R C E  * * * * *
```

1 The separate indicator area attribute, `SI`, is not coded in the `ASSIGN` clause.

2 The Format 2 `COPY` statement, defines data fields and indicators in the record area.

3 Because the file does not have a separate indicator area, response and option indicators are defined in the order in which they are used in the DDS, and the indicator numbers are treated as documentation.

4 All indicators used by the program are defined with meaningful names in data description entries in `WORKING-STORAGE`. Indicator numbers are omitted here because they have no effect. Indicators should be defined in the order needed by the display file.

5 `IN01` in `WORKING-STORAGE` is set on if it is the first day of the month.

6 `FORMAT1` is written to the work station screen:

- The `INDICATORS` phrase causes the contents of the variable `OPTION-INDICS` to be copied to the beginning of the record area.
- Data and indicator values are written to the work station screen.

7 `FORMAT1`, including both data and indicators, is read from the work station screen.

8 The `INDICATORS` phrase causes bytes to be copied from the beginning of the record area to `RESPONSE-INDICS`.

9 If `F` key 5 has been pressed, a program call is processed.

Figure 26 (Part 2 of 2). Example of a Program Using Indicators in the Record Area and the INDICATORS phrase in the I-O Statements

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7784-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Not (N)	Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (b/I/N/K/R/O)	Reserved	Name	Reference (R)	Length	Data Type (b/A/P/S/B/A/B/X/Y/N/I/U/W)	Decimal Positions	Usage (b/O/I/B/H/M)	Location: Line	Pos	Functions
1 2 3 4 5	6	7	8	9 10	11 12	13 14	15 16	17 18	19		20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
	A	*				DISPLAY FILE DDS					FOR INDICATOR EXAMPLES								
	A	*																	
	A																	1	INDARA
	A								R		FORMAT1								CF01(99 'END OF PROGRAM')
	A																		CF05(51 'DAILY REPORT')
	A																		CF09(52 'MONTHLY REPORT')
	A	*																	
	A																10	10	'DEPARTMENT NUMBER: '
	A										DEPTNO		5		I		10	32	
	A																20	26	'PRODUCE MONTHLY REPORTS'
	A				01														DSPATR(BL)
	A	*																	
	A																24	01	'F5 = DAILY REPORT'
	A																24	26	'F9 = MONTHLY REPORT'
	A																24	53	'F1 = TERMINATE'
	A																		
	A																		
	A																		

1 The INDARA keyword is specified, indicators are stored in a separate indicator area, not in the record area. Except for this specification, the DDS for this file is the same as that shown in Figure 24 on page 97.

Figure 27. Example of a Program Using Indicators in a Separate Indicator Area, defined in WORKING-STORAGE by using the Format 2 COPY Statement

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000110 PROGRAM-ID.     EXMPLE717.
       000120*    EXAMPLE PROGRAM - FILE WITH SEPARATE INDICATORS AREA
    3  000130 AUTHOR.       PROGRAMMER NAME.
    4  000140 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
    5  000150 DATE-WRITTEN. 08/30/88.
    6  000160 DATE-COMPILED.  08/31/88 09:31:39   .
    7  000170 ENVIRONMENT DIVISION.
    8  000180 CONFIGURATION SECTION.
    9  000190 SOURCE-COMPUTER. IBM-S38.
   10  000200 OBJECT-COMPUTER. IBM-S38.
   11  000210 INPUT-OUTPUT SECTION.
   12  000220 FILE-CONTROL.
   13  000230     SELECT DISPFILE
   14  000240         ASSIGN TO WORKSTATION-DISPFILE-SI _
   15  000250         ORGANIZATION IS TRANSACTION
   16  000260         ACCESS IS SEQUENTIAL.
   17  000270
   18  000280 DATA DIVISION.
   19  000290 FILE SECTION.
   20  000300 FD  DISPFILE
   21  000310     LABEL RECORDS ARE OMITTED
   22  000320     DATA RECORD IS DISP-REC.
   23  000330 01  DISP-REC.
   24  000340     COPY DDS-ALL-FORMATS OF DISPFILE. __
   25 +000001       05  DISPFILE-RECORD PIC X(5).                                  <-ALL-FMTS
      +000002*  INPUT FORMAT:FORMAT1    FROM FILE DISPFILE   OF LIBRARY EXMPLIB    <-ALL-FMTS
      +000003*                                                                     <-ALL-FMTS
   26 +000004       05  FORMAT1-I     REDEFINES DISPFILE-RECORD.                  <-ALL-FMTS
   27 +000005           06 DEPTNO           PIC X(5).                             <-ALL-FMTS
      +000006* OUTPUT FORMAT:FORMAT1    FROM FILE DISPFILE   OF LIBRARY EXMPLIB    <-ALL-FMTS
      +000007*                                                                     <-ALL-FMTS
      +000008*      05  FORMAT1-O     REDEFINES DISPFILE-RECORD.                  <-ALL-FMTS
   28  000350
   29  000360 WORKING-STORAGE SECTION.
   30  000370 01  CURRENT-DATE.
   31  000380    05  CURR-YEAR                PIC 9(2).
   32  000390    05  CURR-MONTH               PIC 9(2).
   33  000400    05  CURR-DAY                 PIC 9(2).
   34  000410
   35  000420    77 IND-OFF                   PIC 1 VALUE B"0".
   36  000430    77 IND-ON                    PIC 1 VALUE B"1".
   37  000440 01  DISPFILE-INDICS.
   38  000450     COPY DDS-ALL-FORMATS-INDIC OF DISPFILE. ___
   39 +000001       05  DISPFILE-RECORD.                                           <-ALL-FMTS
      +000002*  INPUT FORMAT:FORMAT1    FROM FILE DISPFILE   OF LIBRARY EXMPLIB    <-ALL-FMTS
      +000003*                                                                     <-ALL-FMTS
   40 +000004           06 FORMAT1-I-INDIC.                                       <-ALL-FMTS
   41 +000005                07 IN51         PIC 1  INDIC 51. ____                <-ALL-FMTS
      +000006*                           DAILY REPORT                            <-ALL-FMTS
   42 +000007                07 IN52         PIC 1  INDIC 52.                    <-ALL-FMTS
      +000008*                           MONTHLY REPORT                          <-ALL-FMTS
   43 +000009                07 IN99         PIC 1  INDIC 99.                    <-ALL-FMTS
      +000010*                           END OF PROGRAM                          <-ALL-FMTS
      +000011* OUTPUT FORMAT:FORMAT1    FROM FILE DISPFILE   OF LIBRARY EXMPLIB    <-ALL-FMTS
```

Figure 28 (Part 1 of 2). Example of a Program Using Indicators in a Separate Indicator Area, defined in WORKING-STORAGE by using the Format 2 COPY Statement

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
     +000012*                                                                             <-ALL-FMTS
   44 +000013           06 FORMAT1-O-INDIC.                                               <-ALL-FMTS
   45 +000014                07 IN01         PIC 1  INDIC 01.                           <-ALL-FMTS
   46  000460
   47  000470 PROCEDURE DIVISION.
       000480
       000490 MAIN-PROCESS.
       000500
   48  000510     OPEN I-O DISPFILE.
   49  000520     ACCEPT CURRENT-DATE FROM DATE.
   50  000530     MOVE IND-OFF TO IN99 IN FORMAT1-I-INDIC.
   51  000540     PERFORM DISPLAY-SCREEN THRU READ-AND-PROCESS-SCREEN
       000550         UNTIL IN99 IN FORMAT1-I-INDIC = IND-ON.
   52  000560     CLOSE DISPFILE.
   53  000570     STOP RUN.
       000580
       000590 DISPLAY-SCREEN.
       000600
   54  000610     MOVE ZEROS TO FORMAT1-O-INDIC.
   55  000620     IF CURR-DAY = 01 THEN
   56  000630 _____       MOVE IND-ON TO IN01 IN FORMAT1-O-INDIC.
   57  000640     WRITE DISP-REC FORMAT IS "FORMAT1"
       000650                    INDICATORS ARE FORMAT1-O-INDIC.  6
       000660
       000670 READ-AND-PROCESS-SCREEN.
       000680
   58  000690     MOVE ZEROS TO FORMAT1-I-INDIC.
   59  000700     READ DISPFILE FORMAT IS "FORMAT1"
       000710                   INDICATORS ARE FORMAT1-I-INDIC.  7
   60  000720     IF IN51 IN FORMAT1-I-INDIC = IND-ON THEN
   61  000730  8        CALL "DAILY" USING DEPTNO
       000740     ELSE
   62  000750         IF IN52 IN FORMAT1-I-INDIC = IND-ON THEN
   63  000760             CALL "MONTHLY" USING DEPTNO.
                        * * * * *  E N D  O F  S O U R C E  * * * * *
```

1 The separate indicator area attribute, `SI`, is specified in the `ASSIGN` clause.

2 The Format 2 `COPY` statement, generates data descriptions in the record area for data fields only. The data description entries for the indicators are not generated because a separate indicator area has been specified for the file.

3 The Format 2 `COPY` statement, with the `INDICATOR` attribute, `INDIC`, defines data description entries in `WORKING-STORAGE` for all indicators used in the `DDS` for the record format for the file.

4 Because the file has a separate indicator area, the indicator numbers used in the data description entries are not treated as documentation.

5 `IN01` in the separate indicator area for `FORMAT1` is set on if it is the first day of the month.

6 The `INDICATORS` phrase is required to send indicator values to the work station screen.

7 The `INDICATORS` phrase is required to receive indicator values from the work station screen. If `F` key 5 has been pressed, `IN51` is set on.

8 If `IN51` has been set on, a program call is processed.

Figure 28 (Part 2 of 2). Example of a Program Using Indicators in a Separate Indicator Area, defined in WORKING-STORAGE by using the Format 2 COPY Statement

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.     EXMPLE720.
       000300*    EXAMPLE PROGRAM
       000400*    FILE WITH SEPARATE INDICATORS AREA IN WORKING STORAGE
    3  000500 AUTHOR.       PROGRAMMER NAME.
    4  000600 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
    5  000700 DATE-WRITTEN. 08/30/88.
    6  000800 DATE-COMPILED.  08/31/88 11:21:24   .
    7  000900 ENVIRONMENT DIVISION.
    8  001000 CONFIGURATION SECTION.
    9  001100 SOURCE-COMPUTER. IBM-S38.
   10  001200 OBJECT-COMPUTER. IBM-S38.
   11  001300 INPUT-OUTPUT SECTION.
   12  001400 FILE-CONTROL.
   13  001500     SELECT DISPFILE
   14  001600         ASSIGN TO WORKSTATION-DISPFILE-SI _
   15  001700         ORGANIZATION IS TRANSACTION
   16  001800         ACCESS IS SEQUENTIAL.
   17  001900
   18  002000 DATA DIVISION.
   19  002100 FILE SECTION.
   20  002200 FD  DISPFILE
   21  002300     LABEL RECORDS ARE OMITTED
   22  002400     DATA RECORD IS DISP-REC.
   23  002500 01  DISP-REC.
   24  002600     COPY DDS-ALL-FORMATS OF DISPFILE. __
   25 +000001       05  DISPFILE-RECORD PIC X(5).                                    <-ALL-FMTS
      +000002*  INPUT FORMAT:FORMAT1    FROM FILE DISPFILE   OF LIBRARY EXMPLIB      <-ALL-FMTS
      +000003*                                                                       <-ALL-FMTS
   26 +000004       05  FORMAT1-I     REDEFINES DISPFILE-RECORD.                    <-ALL-FMTS
   27 +000005           06 DEPTNO            PIC X(5).                             <-ALL-FMTS
      +000006* OUTPUT FORMAT:FORMAT1    FROM FILE DISPFILE   OF LIBRARY EXMPLIB      <-ALL-FMTS
      +000007*                                                                       <-ALL-FMTS
      +000008*      05  FORMAT1-O     REDEFINES DISPFILE-RECORD.                    <-ALL-FMTS
   28  002700
   29  002800 WORKING-STORAGE SECTION.
   30  002900 01  CURRENT-DATE.
   31  003000    05  CURR-YEAR                PIC 9(2).
   32  003100    05  CURR-MONTH               PIC 9(2).
   33  003200    05  CURR-DAY                 PIC 9(2).
   34  003300
   35  003400 01  INDIC-AREA.
   36  003500    05 INDIC-TABLE  OCCURS 99     PIC 1    INDICATOR 1. ___
   37  003600       88  IND-OFF                         VALUE B"0".
   38  003700       88  IND-ON                          VALUE B"1".
   39  003800
   40  003900 01  DISPFLIE-INDIC-USAGE.
   41  004000    05  IND-NEW-MONTH            PIC 9(2) VALUE 01.
   42  004100    05  IND-DAILY                PIC 9(2) VALUE 51.
   43  004200 ____  05  IND-MONTHLY              PIC 9(2) VALUE 52.
   44  004300    05  IND-EOJ                  PIC 9(2) VALUE 99.
   45  004400
   46  004500 PROCEDURE DIVISION.
       004600
       004700 EXMPLE-MAIN.
```

Figure 29 (Part 1 of 2). Example of a Program Using Indicators in a Separate Indicator Area, Defined in a Table in WORKING-STORAGE

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      004800
   47 004900     OPEN I-O DISPFILE.
   48 005000     ACCEPT CURRENT-DATE FROM DATE.
   49 005100     SET IND-OFF (IND-EOJ) TO TRUE.
   50 005200     PERFORM DISPLAY-SCREEN THRU READ-AND-PROCESS-SCREEN
      005300         UNTIL IND-ON (IND-EOJ).
   51 005400     CLOSE DISPFILE.
   52 005500     STOP RUN.
      005600
      005700 DISPLAY-SCREEN.
      005800
   53 005900     MOVE ZEROS TO INDIC-AREA.
   54 006000     IF CURR-DAY = 01 THEN
   55 006100        SET IND-ON (IND-NEW-MONTH) TO TRUE. _____
   56 006200     WRITE DISP-REC FORMAT IS "FORMAT1"
      006300                    INDICATORS ARE INDIC-TABLE.  6
      006400
      006500 READ-AND-PROCESS-SCREEN.
      006600
   57 006700     READ DISPFILE FORMAT IS "FORMAT1"
      006800                   INDICATORS ARE INDIC-TABLE.  7
   58 006900  8    IF IND-ON (IND-DAILY) THEN
   59 007000         CALL "DAILY" USING DEPTNO
      007100     ELSE
   60 007200         IF IND-ON (IND-MONTHLY) THEN
   61 007300             CALL "MONTHLY" USING DEPTNO.
                         * * * * *  E N D  O F  S O U R C E  * * * * *
```

1 The separate indicator area attribute, SI, is specified in the ASSIGN clause.

2 The Format 2 COPY statement, generates fields in the record area for data fields only.

3 A table of 99 Boolean data items is defined in WORKING-STORAGE. The INDICATOR clause for this data description entry causes these data items to be associated with indicators 1 through 99 respectively. The use of such a table may result in improved performance as compared to the use of a group item with multiple subordinate entries for individual indicators; however, you must consider the number of references and indicators for example, to realize improved performance.

4 A series of data items is defined in WORKING-STORAGE to provide meaningful subscript names with which to refer to the table of indicators. The use of such data items is not required.

5 INDIC-TABLE (01) in the separate indicator area for FORMAT1 is set on if it is the first day of the month.

6 The INDICATOR phrase is required to send indicator values to the work station screen.

7 The INDICATOR phrase is required to receive indicator values from the work station screen. If F key 5 has been pressed, INDIC-TABLE (51) is set on.

8 If INDIC-TABLE (51) has been set on, a program call is processed.

Figure 29 (Part 2 of 2). Example of a Program Using Indicators in a Separate Indicator Area, Defined in a Table in WORKING-STORAGE

Subfiles

Subfiles can be specified in the DDS for a display file or mixed file to allow the user to handle multiple records of the same type on a display (see Figure 30 on page 107). A subfile is a group of records that is read from or written to a display device. For example, a program reads records from a data base file and creates a subfile of output records. When the entire subfile has been written, the program sends the entire subfile to a display device in one write operation. The work station user can change data or enter additional data in the subfile; the program then

reads the entire subfile from the display device into the program and processes each record in the subfile individually.

Records to be included in a subfile are specified in the DDS for the file. The number of records that can be contained in a subfile must also be specified in the DDS. One file can contain more than one subfile; however, only twelve subfiles can be active concurrently for a device. Twelve subfiles can be displayed on a device at the same time.

The DDS for a subfile consists of two record formats: a subfile record format and a subfile control record format. The subfile record format contains the field descriptions of all the records in the subfile. Specification of the subfile control record format on the `READ` or `WRITE` statement causes the physical read, write, or setup operations of a subfile to take place. Figure 31 on page 109 shows an example of the DDS for a subfile record format, and Figure 32 on page 111 shows an example of the DDS for a subfile control record format.

Note: In a mixed file, the formats used for defining the subfile records and the subfile control record can be used only in I-O operations to display devices.

```
Customer Name Search

Search Code  _____

Number Name                  Address               City                  State

XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
XXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXXXXXX  XX
```

Figure 30. Subfile Display

To use a subfile for a display file or mixed file in a COBOL program, the `SUBFILE` phrase must be specified with the input/output operation. The valid subfile operations are:

- `READ SUBFILE` file-name `RECORD`
- `WRITE SUBFILE` record-name
- `REWRITE SUBFILE` record-name.

In COBOL, subfiles can be processed sequentially with the `READ SUBFILE NEXT MODIFIED` statement, or processed randomly by specifying a relative key value.

The TRANSACTION file must be an externally described file. In COBOL, all access to the subfile is done with a relative record number. If the SUBFILE phrases are used with a TRANSACTION file, the SELECT statement in the Environment Division must state that ACCESS MODE IS DYNAMIC and must specify the RELATIVE KEY to be used.

If more than one display device is acquired by a display file or mixed file, there is a separate subfile for each individual display device. If a subfile has been created for a particular display device acquired by a TRANSACTION file, all input operations that refer to a record format for the subfile are processed against the subfile belonging to that device. See the discussions on the TERMINAL phrase later in this chapter for information about how to determine which device is used. Any operations that reference a record format name that is not designated as a subfile are processed as an input/output operation directly to the display device.

Use of Subfiles

Some typical uses of subfiles include:

- ***Display only***. The work station user reviews the display.
- ***Display with selection***. The user requests more information about one of the items on the display.
- ***Modification***. The user modifies one or more of the records.
- ***Input only, with no validity checking***. A subfile is used for a data entry function.
- ***Input only, with validity checking***. A subfile is used for a data entry function, but the records are checked.
- ***Combination of tasks***. A subfile can be used for a display with modification.

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7764-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (b/R/K/S/C)	Reserved	Name	Reference (R)	Length	Data Type (b A/P/S/B A/S/X/Y/N/I/M)	Decimal Positions	Usage (b/O/I/B/H/M)	Location: Line	Location: Pos	Functions
	A	*						DDS	F		OR THE DISPLAY DEVICE FILE								ARC010D
	A	*									ACCOUNTS RECEIVABLE INTERACTIVE PAYMENT UPDATE								
	A	*																	
	A								R		SUBFILE1							1	SFL
	A																		TEXT('SUBFILE FOR CUSTOMER PAYMENT')
	A	*																	
	A										ACPPMT		4	A		I	5	4	TEXT('ACCEPT PAYMENT')
	A																2		VALUES('*YES' '*NO') 3
	A		51																DSPATR(RI MDT)
	A		N51																DSPATR(ND PR)
	A	*																	
	A										CUST		5			B	5	15	TEXT('CUSTOMER NUMBER')
	A		52														4		DSPATR(RI)
	A		53																DSPATR(ND)
	A		54																DSPATR(PR)
	A	*																	
	A										AMPAID		8	02		B	5	24	TEXT('AMOUNT PAID')
	A																		CHECK(FE) 5
	A																		AUTO(RAB) 6
	A																		CMP(GT 0) 7
	A		52																DSPATR(RI)
	A		53																DSPATR(ND)
	A		54																DSPATR(PR)
	A	*																	
	A										ECPMSG		31	A		O	5	37	TEXT('EXCEPTION MESSAGE')
	A		52																DSPATR(RI)
	A		53																DSPATR(ND)
	A		54																DSPATR(BL)
	A	*																	
	A										OVRPMT		8	Y	2	O	5	70	TEXT('OVER PAYMENT')
	A																	8	EDTCDE(1)
	A		55																DSPATR(BL) 9
	A		56																DSPATR(ND)
	A	*																	
	A										STSCDE		1	A		H			TEXT('STATUS CODE')

Figure 31 (Part 1 of 2). Data Description Specifications for a Subfile Record Format

The data description specifications (DDS) for a subfile record format describe the records in the subfile:

1 The SFL keyword identifies the record format as a subfile.

2 The line and position entries define the location of the fields on the display.

3 The VALUES keyword specifies that the user can only specify *YES or *NO as values for the ACPPMT field.

4 The usage entries define whether the named field is to be an output (O), input (I), output/input (B), or hidden (H) field.

5 The entry CHECK(FE) specifies that the user cannot skip to the next input field without pressing one of the field exit keys.

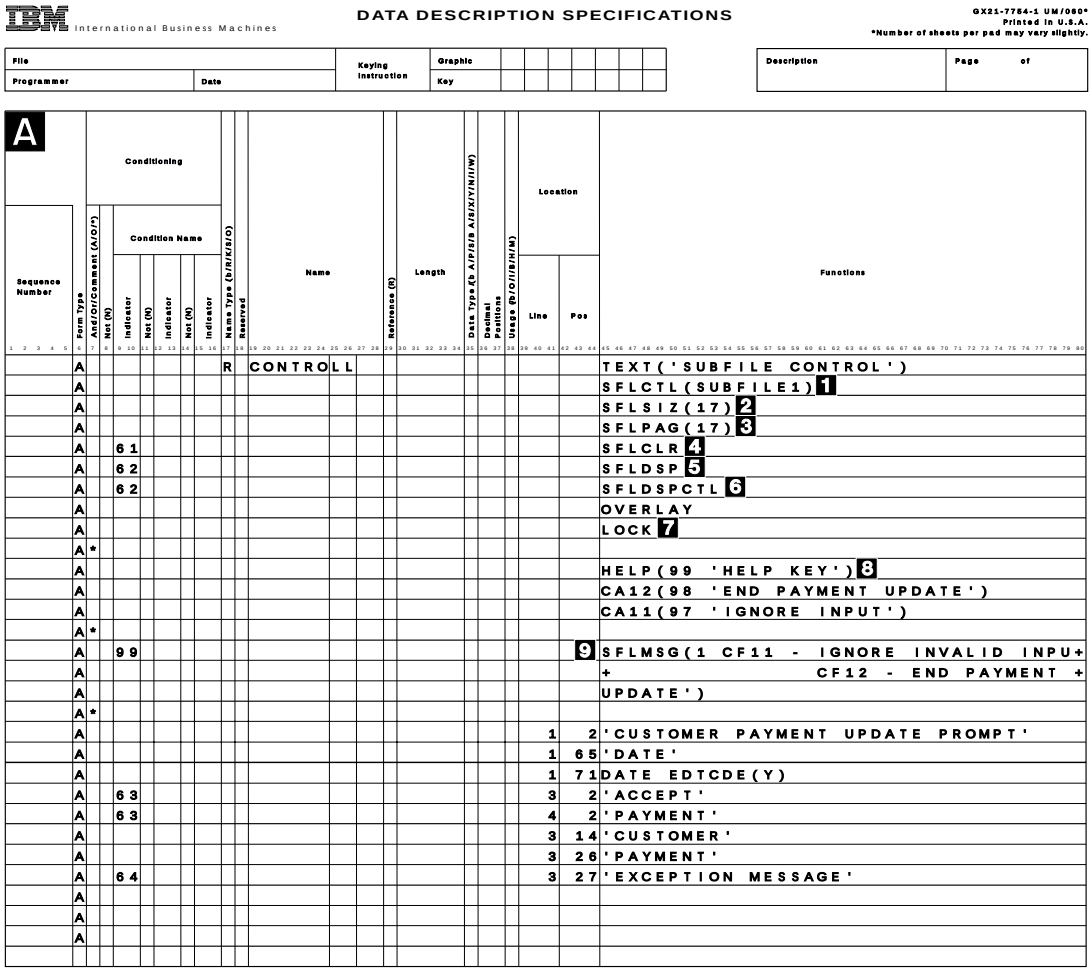
6 The entry AUTO(RAB) specifies that data entered into the field AMPAID is to be automatically right-justified, and the leading characters are to be filled with blanks.

7 The entry CMP(GT 0) specifies that the data entered for the field AMPAID is to be compared to zero to ensure that the value is greater than zero.

8 The EDTCDE keyword specifies the desired editing for output field OVRPMT. EDTCDE(1) indicates that the field OVRPMT is to be printed with commas, decimal point, and no sign. Also, a zero balance will be printed and leading zeros will be suppressed.

9 The DSPATR keyword is used to specify the display attributes for the named field when the corresponding indicator status is true. The attributes specified are BL (blink), RI (reverse image), PR (protect), MDT (set modified data tag), and ND (nondisplay).

Figure 31 (Part 2 of 2). Data Description Specifications for a Subfile Record Format



IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7754-1 UM/080*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Name Type (b/R/K/S/C/)	Reserved	Name	Reference (R)	Length	Data Type (b A/P/S/B A/S/X/Y/N/I/M/W)	Decimal Positions	Usage (b/O/I/B/H/M)	Location: Line	Location: Pos	Functions
	A								R		CONTROLL								TEXT('SUBFILE CONTROL')
	A																		SFLCTL(SUBFILE1) 1
	A																		SFLSIZ(17) 2
	A																		SFLPAG(17) 3
	A			61															SFLCLR 4
	A			62															SFLDSP 5
	A			62															SFLDSPCTL 6
	A																		OVERLAY
	A																		LOCK 7
	A	*																	
	A																		HELP(99 'HELP KEY') 8
	A																		CA12(98 'END PAYMENT UPDATE')
	A																		CA11(97 'IGNORE INPUT')
	A	*																	
	A			99														9	SFLMSG(1 CF11 - IGNORE INVALID INPU+
	A																		+ CF12 - END PAYMENT +
	A																		UPDATE')
	A	*																	
	A																1	2	'CUSTOMER PAYMENT UPDATE PROMPT'
	A																1	65	'DATE'
	A																1	71	DATE EDTCDE(Y)
	A			63													3	2	'ACCEPT'
	A			63													4	2	'PAYMENT'
	A																3	14	'CUSTOMER'
	A																3	26	'PAYMENT'
	A			64													3	27	'EXCEPTION MESSAGE'
	A																		
	A																		
	A																		

The subfile control record format defines the attributes of the subfile, the search input field, constants, and command keys. The keywords used indicate the following:

1 SFLCTL identifies this record as a subfile control record and names the associated subfile record (SUBFILE1).

2 SFLSIZ indicates the total number of records to be included in the subfile (17).

3 SFLPAG indicates the total number of records in a page (17).

4 SFLCLR indicates when the subfile should be cleared (when indicator 61 is on).

5 SFLDSP indicates when to display the subfile (when indicator 62 is on).

6 SFLDSPCTL indicates when to display the subfile control record (when indicator 62 is on).

7 The LOCK keyword prevents the work station user from using the keyboard when the CONTROL1 record format is initially displayed.

8 HELP allows the user to press the Help key and sets indicator 99 on.

9 SFLMSG identifies the constant as a message that is displayed if indicator 99 is on.

In addition to the control information, the subfile control record format also defines the constants to be used as column headings for the subfile record format.

Figure 32. Data Description Specifications for a Subfile Control Record Format

Multiple Device Files and Single Device Files

A multiple device file is either a display file or a mixed file capable of having more than one program device acquired for that file. Communications and BSC files can never be defined as multiple device files.

A display file is capable of having multiple program devices when the MAXDEV parameter of the CRTDSPF CL command is greater than 1.

A mixed file is capable of having multiple program devices when the MAXPGMDEV parameter of the CRTMXDF CL command is greater than 1. Once the mixed file has been created, the required devices must be added to it by means of the ADDDSPDEVE, the ADDCMNDEVE, and/or the ADDBSCDEVE CL commands.

COBOL determines at run time whether a file is a single device file or a multiple device file based on whether the file is *capable* of having multiple devices. The actual number of devices acquired does not affect whether a file is considered a single or multiple device file. Determination of whether a file is a single or a multiple device file is *not* done at compilation time based on the current description of the display or mixed file.

For multiple device files, if a particular program device is to be used in an I-O statement, that device is specified by the TERMINAL phrase. The TERMINAL phrase can also be specified for a single device file.

The following example illustrates the use of multiple device files. The program uses a mixed file, and is intended to be run in batch mode. The program acquires terminals and invites those terminals via a sign-on screen. After the terminals are invited, they are polled. If nobody signs on before the wait time expires, the program ends. If the user enters a valid password, he is allowed to update an employee file by calling another COBOL program. Once the update is complete, the device is invited again and the terminals are polled again.

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7754-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Not (N)	Condition Name: indicator	Not (N)	indicator	Not (N)	indicator	Name Type (R/K/S/C)	Reserved	Name	Reference (R)	Length	Data Type (B-A/P/B/S A/N/J/Y/N/I/W)	Decimal Positions	Usage (B/O/I/H/M)	Location: Line	Pos	Functions
	A	*																	
	A	*	DDS FOR THE MIXED FILE MULT																
	A	*																	
	A								R		SIGNON								INVITE 1
	A															O	5	20	'▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒'
	A																		DSPATR(RI)
	A															O	6	20	'▒▒'
	A																		DSPATR(RI)
																O	6	38	'▒▒'
																			DSPATR(RI)
																O	7	20	'▒▒'
																			DSPATR(RI)
																O	7	27	'M D F'
																			DSPATR(HI BL)
																O	7	38	'▒▒'
																			DSPATR(RI)
																O	8	20	'▒▒'
																			DSPATR(RI)
																O	8	38	'▒▒'
	A																		DSPATR(RI)
	A															O	9	20	'▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒▒'
	A																		DSPATR(RI)
	A															O	20	20	'PLEASE LOGON'
	A																		DSPATR(HI)
	A										PASSWORD		10	A		I	20	43	DSPATR(PC ND)
	A										WRONG		20	A		O	21	43	
	A					R					UPDATE								
	A															O	3	5	'UPDATE OF PERSONNEL FILE'
	A																		DSPATR(BL)
	A															O	7	5	'TYPE IN EMPLOYEE NUMBER +
	A																		TO BE UPDATED'
	A										NUM		7	A		I	7	44	DSPATR(RI PC)
	A					R					EMPLOYEE								
	A															O	3	5	'EMPLOYEE NUMBER'
	A										NUM		7	A		B	3	25	DSPATR(PC)
	A															O	5	5	'EMPLOYEE NAME'
	A										NAME		30	A		B	5	25	DSPATR(PC)
	A															O	7	5	'EMPLOYEE ADDRESS'
	A															O	9	5	'STREET'
	A										STREET		30	A		B	9	25	DSPATR(PC)
	A															O	11	5	'APARTMENT NUMBER'
	A										APTNO		5	A		B	11	25	DSPATR(PC)
	A															O	13	5	'CITY'
	A										CITY		20	A		B	13	25	DSPATR(PC)
	A															O	15	5	'PROVINCE'
	A										PROV		20	A		B	15	25	DSPATR(PC)
	A					R					RECOVERY								
	A															O	3	5	'THE EMPLOYEE NUMBER YOU +
	A																		HAVE GIVEN IS INVALID'
	A															O	6	5	'TYPE Y TO RETRY'
	A															O	8	5	'TYPE N TO EXIT'
	A										ANSWER		1	X		I	10	5	DSPATR(RI PC)
	A																		VALUES('Y' 'N')
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		

1 The format SIGNON has the keyword INVITE associated with it. This means that, if format SIGNON is used in a WRITE statement, the device to which it is writing will be invited.

Figure 33 (Part 1 of 10). Example of the Use of Multiple Device Files

IBM International Business Machines — DATA DESCRIPTION SPECIFICATIONS — GX21-7764-1 UM/060* Printed in U.S.A. *Number of sheets per pad may vary slightly.

File | Keying Instruction: Graphic | Description | Page of
Programmer | Date | Key

A

Sequence Number	Form Type	Conditioning: And/Or/Comment (A/O/*)	Not (N)	Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (S/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (B A/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (B/O/I/B/H/M)	Location: Line	Pos	Functions
	A	*																	
	A	*	DDS FOR THE PHYSICAL FILE PASSWORD																
	A	*																	
	A																		UNIQUE
	A								R		PASSWORDS								
	A										PASSKEY		10						
	A										PASSWORD		10						
	A								K		PASSKEY								
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		

Figure 33 (Part 2 of 10). Example of the Use of Multiple Device Files

IBM International Business Machines — DATA DESCRIPTION SPECIFICATIONS — GX21-7764-1 UM/060* Printed in U.S.A. *Number of sheets per pad may vary slightly.

File | Keying Instruction: Graphic | Description | Page of
Programmer | Date | Key

A

Sequence Number	Form Type	Conditioning: And/Or/Comment (A/O/*)	Not (N)	Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (S/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (B A/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (B/O/I/B/H/M)	Location: Line	Pos	Functions
	A	*																	
	A	*	DDS FOR THE PHYSICAL FILE TERM																
	A	*	WHICH CONTAINS THE LIST OF TERMINALS																
	A	*																	
	A								R		TERM								
	A										TERM		10						
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		

Figure 33 (Part 3 of 10). Example of the Use of Multiple Device Files

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.     EXMPLEMDF.
    3  000300   AUTHOR.       PROGRAMMER NAME.
       000400
       000500*************************************************************
       000600*  THE FOLLOWING PROGRAM DEMONSTRATES SOME OF THE FUNCTIONS  *
       000700*  AVAILABLE WITH MULTIPLE DEVICE FILE SUPPORT.              *
       000800*************************************************************
       000900
    4  001000   INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
    5  001100   DATE-WRITTEN. 08/30/88.
    6  001200   DATE-COMPILED.  08/30/88 14:45:58   .
    7  001300 ENVIRONMENT DIVISION.
    8  001400 CONFIGURATION SECTION.
    9  001500 SOURCE-COMPUTER. IBM-S38.
   10  001600 OBJECT-COMPUTER. IBM-S38.
   11  001700 SPECIAL-NAMES.
   12  001800     ATTRIBUTE-DATA IS ATTR. _
   13  001900 INPUT-OUTPUT SECTION.
   14  002000 FILE-CONTROL.
   15  002100     SELECT MULTIPLE-FILE
   16  002200     ASSIGN TO WORKSTATION-MULT __
   17  002300     ORGANIZATION IS TRANSACTION
   18  002400     ACCESS MODE IS SEQUENTIAL
   19  002500     FILE STATUS IS MULTIPLE-FS1, MULTIPLE-FS2 ___
   20  002600     CONTROL-AREA IS MULTIPLE-CONTROL-AREA. ____
   21  002700
   22  002800     SELECT TERMINAL-FILE
   23  002900     ASSIGN TO DATABASE-TERM
   24  003000     ORGANIZATION IS SEQUENTIAL
   25  003100     ACCESS IS SEQUENTIAL
   26  003200     FILE STATUS IS TERMINAL-FS1.
   27  003300
   28  003400     SELECT PASSWORD-FILE
   29  003500     ASSIGN TO DATABASE-PASSWORD
   30  003600     ORGANIZATION IS INDEXED
   31  003700     RECORD KEY IS EXTERNALLY-DESCRIBED-KEY
   32  003800     ACCESS MODE IS RANDOM
   33  003900     FILE STATUS IS PASSWORD-FS1.
   34  004000
   35  004100     SELECT PRINTER-FILE
   36  004200     ASSIGN TO PRINTER-QPRINT.
   37  004300 DATA DIVISION.
   38  004400 FILE SECTION.
   39  004500 FD  MULTIPLE-FILE.
   40  004600 01  MULTIPLE-REC. COPY DDS-SIGNON OF MULT. _____
   41 +000001       05  MULT-RECORD PIC X(30).                                  SIGNON
      +000002*  INPUT FORMAT:SIGNON     FROM FILE MULT       OF LIBRARY EXMPLIB  SIGNON
      +000003*                                                                  SIGNON
   42 +000004       05  SIGNON-I      REDEFINES MULT-RECORD.                    SIGNON
   43 +000005           06 PASSWORD          PIC X(10).  6                      SIGNON
      +000006* OUTPUT FORMAT:SIGNON     FROM FILE MULT       OF LIBRARY EXMPLIB  SIGNON
      +000007*                                                                  SIGNON
   44 +000008       05  SIGNON-O      REDEFINES MULT-RECORD.                    SIGNON
   45 +000009           06 FILLER            PIC X(10).                         SIGNON
```

Figure 33 (Part 4 of 10). Example of the Use of Multiple Device Files

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   46 +000010           06 WRONG             PIC X(20).                                    SIGNON
   47  004700
   48  004800 FD  TERMINAL-FILE.
   49  004900 01  TERMINAL-REC. COPY DDS-ALL-FORMATS OF TERM.
   50 +000001       05  TERM-RECORD PIC X(10).                                            <-ALL-FMTS
      +000002*    I-O FORMAT:TERM       FROM FILE TERM       OF LIBRARY EXMPLIB          <-ALL-FMTS
      +000003*                                                                           <-ALL-FMTS
   51 +000004       05  TERM          REDEFINES TERM-RECORD.                             <-ALL-FMTS
   52 +000005           06 TERM              PIC X(10).                                    <-ALL-FMTS
   53  005000
   54  005100 FD  PASSWORD-FILE.
   55  005200 01  PASSWORD-REC. COPY DDS-ALL-FORMATS OF PASSWORD.
   56 +000001       05  PASSWORD-RECORD PIC X(20).                                        <-ALL-FMTS
      +000002*    I-O FORMAT:PASSWORDS  FROM FILE PASSWORD   OF LIBRARY EXMPLIB          <-ALL-FMTS
      +000003*                                                                           <-ALL-FMTS
      +000004*THE KEY DEFINITIONS FOR RECORD FORMAT  PASSWORDS                          <-ALL-FMTS
      +000005*  NUMBER              NAME               RETRIEVAL     TYPE    ALTSEQ     <-ALL-FMTS
      +000006*   0001   PASSKEY                         ASCENDING      AN      NO       <-ALL-FMTS
   57 +000007       05  PASSWORDS     REDEFINES PASSWORD-RECORD.                         <-ALL-FMTS
   58 +000008           06 PASSKEY           PIC X(10).                                    <-ALL-FMTS
   59 +000009           06 PASSWORD          PIC X(10).                                    <-ALL-FMTS
   60  005300
   61  005400 FD  PRINTER-FILE.
   62  005500 01  PRINTER-REC.
   63  005600    05 PRINTER-RECORD        PIC X(132).
   64  005700
   65  005800 WORKING-STORAGE SECTION.
   66  005900
       006000*****************************************************************
       006100*       DECLARE THE FILE STATUS FOR EACH FILE                   *
       006200*****************************************************************
   67  006300
   68  006400 01  MULTIPLE-FS1            PIC X(2)    VALUE SPACES.
   69  006500 01  MULTIPLE-FS2. 7
   70  006600    05  MULTIPLE-MAJOR       PIC X(2)    VALUE SPACES.
   71  006700    05  MULTIPLE-MINOR       PIC X(2)    VALUE SPACES.
   72  006800 01  TERMINAL-FS1            PIC X(2)    VALUE SPACES.
   73  006900 01  PASSWORD-FS1            PIC X(2)    VALUE SPACES.
   74  007000
       007100*****************************************************************
       007200*       DECLARE STRUCTURE FOR HOLDING FILE ATTRIBUTES           *
       007300*****************************************************************
   75  007400
   76  007500 01  STATION-ATTR.
   77  007600    05 STATION-TYPE          PIC X(1). 8
   78  007700    05 STATION-SIZE          PIC X(1).
   79  007800    05 STATION-LOC           PIC X(1).
   80  007900    05 FILLER                PIC X(1).
   81  008000    05 STATION-ACQUIRE       PIC X(1).
   82  008100    05 STATION-INVITE        PIC X(1).
   83  008200    05 STATION-DATA          PIC X(1).
   84  008300    05 STATION-STATUS        PIC X(1).
   85  008400    05 STATION-DISPLAY       PIC X(1).
   86  008500    05 STATION-KEYBOARD      PIC X(1).
   87  008600    05 STATION-SIGNON        PIC X(1).
```

Figure 33 (Part 5 of 10). Example of the Use of Multiple Device Files

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   88  008700    05 FILLER               PIC X(5).
   89  008800
       008900*************************************************************
       009000*       DECLARE THE CONTROL AREA FOR MULTIPLE-FILE           *
       009100*************************************************************
   90  009200
   91  009300 01  MULTIPLE-CONTROL-AREA.
   92  009400    05 MULTIPLE-KEY-FEEDBACK PIC X(2)    VALUE SPACES.
   93  009500    05 MULTIPLE-DEVICE-NAME  PIC X(10)   VALUE SPACES.
   94  009600    05 MULTIPLE-FORMAT-NAME  PIC X(10)   VALUE SPACES.
   95  009700
       009800*************************************************************
       009900*               DECLARE ERROR REPORT VARIABLES              *
       010000*************************************************************
   96  010100
   97  010200 01  HEADER-LINE.
   98  010300    05 FILLER                PIC X(60)   VALUE SPACES.
   99  010400    05 FILLER                PIC X(72)
  100  010500                             VALUE "MDF ERROR REPORT".
  101  010600 01  DETAIL-LINE.
  102  010700    05 FILLER                PIC X(15)   VALUE SPACES.
  103  010800    05 DESCRIPTION           PIC X(25)   VALUE SPACES.
  104  010900    05 DETAIL-VALUE          PIC X(92)   VALUE SPACES.
  105  011000
       011100*************************************************************
       011200*        DECLARE COUNTERS, FLAGS AND STORAGE VARIABLES       *
       011300*************************************************************
  106  011400
  107  011500 01  CURRENT-TERMINAL        PIC X(10)   VALUE SPACES.
  108  011600 01  TERMINAL-ARRAY.
  109  011700    05 LIST-OF-TERMINALS OCCURS 250 TIMES.
  110  011800      07 DEVICE-NAME         PIC X(10).
  111  011900 01  COUNTER                 PIC 9(3)    VALUE IS 1.
  112  012000 01  NO-OF-TERMINALS         PIC 9(3)    VALUE IS 1.
  113  012100 01  TERMINAL-LIST-FLAG      PIC 1.
  114  012200    88 END-OF-TERMINAL-LIST              VALUE IS B"1".
  115  012300    88 NOT-END-OF-TERMINAL-LIST          VALUE IS B"0".
  116  012400 01  NO-DATA-FLAG            PIC 1.
  117  012500    88 NO-DATA-AVAILABLE                 VALUE IS B"1".
  118  012600    88 DATA-AVAILABLE                    VALUE IS B"0".
  119  012700
  120  012800 PROCEDURE DIVISION.
       012900
       013000 DECLARATIVES.
       013100
       013200 MULTIPLE-SECTION SECTION.
```

Figure 33 (Part 6 of 10). Example of the Use of Multiple Device Files

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
       013300     USE AFTER STANDARD EXCEPTION PROCEDURE ON MULTIPLE-FILE.
       013400
       013500 MULTIPLE-PARAGRAPH.
  121  013600     WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
  122  013700     MOVE "FILE NAME IS:" TO DESCRIPTION OF DETAIL-LINE.
  123  013800     MOVE "MULTIPLE FILE" TO DETAIL-VALUE OF DETAIL-LINE.
  124  013900     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
  125  014000     MOVE "FILE STATUS IS:" TO DESCRIPTION OF DETAIL-LINE.
  126  014100     MOVE MULTIPLE-FS1 TO DETAIL-VALUE OF DETAIL-LINE.
  127  014200     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
  128  014300     MOVE "EXTENDED STATUS IS:" TO DESCRIPTION OF DETAIL-LINE. _________
  129  014400     MOVE MULTIPLE-FS2 TO DETAIL-VALUE OF DETAIL-LINE.
  130  014500     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
  131  014600     ACCEPT STATION-ATTR FROM ATTR.
  132  014700     MOVE "FILE ATTRIBUTES ARE:" TO DESCRIPTION OF DETAIL-LINE.  9A
  133  014800     MOVE STATION-ATTR TO DETAIL-VALUE OF DETAIL-LINE.
  134  014900     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
  135  015000     STOP RUN.
       015100
       015200 TERMINAL-SECTION SECTION.
       015300     USE AFTER STANDARD EXCEPTION PROCEDURE ON TERMINAL-FILE.
       015400 TERMINAL-PARAGRAPH.
  136  015500     WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
  137  015600     MOVE "FILE NAME IS:" TO DESCRIPTION OF DETAIL-LINE.
  138  015700     MOVE "TERMINAL FILE" TO DETAIL-VALUE OF DETAIL-LINE.
  139  015800     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
  140  015900     MOVE "FILE STATUS IS:" TO DESCRIPTION OF DETAIL-LINE.
  141  016000     MOVE TERMINAL-FS1 TO DETAIL-VALUE OF DETAIL-LINE.
  142  016100     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
  143  016200     STOP RUN.
       016300
       016400 PASSWORD-SECTION SECTION.
       016500     USE AFTER STANDARD EXCEPTION PROCEDURE ON PASSWORD-FILE.
       016600 PASSWORD-PARAGRAPH.
  144  016700     WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
  145  016800     MOVE "FILE NAME IS:" TO DESCRIPTION OF DETAIL-LINE.
  146  016900     MOVE "PASSWORD FILE" TO DETAIL-VALUE OF DETAIL-LINE.
  147  017000     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
  148  017100     MOVE "FILE STATUS IS:" TO DESCRIPTION OF DETAIL-LINE.
  149  017200     MOVE PASSWORD-FS1 TO DETAIL-VALUE OF DETAIL-LINE.
  150  017300     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
  151  017400     STOP RUN.
       017500
```

Figure 33 (Part 7 of 10). Example of the Use of Multiple Device Files

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      017600 END DECLARATIVES.
      017700
      017800*************************************************************
      017900*        MAIN PROGRAM LOGIC BEGINS HERE                     *
      018000*************************************************************
      018100
      018200 MAIN-LINE SECTION.
      018300 MAIN-LINE-PARAGRAPH.
  152 018400     OPEN I-O    MULTIPLE-FILE
      018500          INPUT  TERMINAL-FILE
      018600          I-O    PASSWORD-FILE  10
      018700          OUTPUT PRINTER-FILE.
      018800
  153 018900       MOVE 1 TO COUNTER.
  154 019000       SET NOT-END-OF-TERMINAL-LIST TO TRUE.
      019100       PERFORM
  155 019200           FILL-TERMINAL-LIST UNTIL END-OF-TERMINAL-LIST.
      019300       PERFORM
  156 019400           ACQUIRE-AND-INVITE-TERMINALS
      019500               VARYING COUNTER FROM 1 BY 1
      019600               UNTIL COUNTER GREATER THAN NO-OF-TERMINALS.
  157 019700       MOVE 1 TO COUNTER.
  158 019800       SET DATA-AVAILABLE TO TRUE.
      019900       PERFORM
  159 020000           POLL-TERMINALS UNTIL NO-DATA-AVAILABLE.
      020100       PERFORM
  160 020200           DROP-TERMINALS
      020300               VARYING COUNTER FROM 1 BY 1
      020400               UNTIL COUNTER GREATER THAN NO-OF-TERMINALS.
  161 020500       CLOSE   MULTIPLE-FILE
      020600               TERMINAL-FILE
      020700               PASSWORD-FILE
      020800               PRINTER-FILE.
  162 020900       STOP RUN.
      021000
      021100*************************************************************
      021200*                PROCEDURES                                 *
      021300*************************************************************
      021400
      021500 PROCEDURE-SECTION SECTION.
      021600 FILL-TERMINAL-LIST.
  163 021700     READ TERMINAL-FILE RECORD INTO LIST-OF-TERMINALS(COUNTER)
      021800         AT END
  164 021900             SET END-OF-TERMINAL-LIST TO TRUE
  165 022000             SUBTRACT 1 FROM COUNTER
  166 022100             MOVE COUNTER TO NO-OF-TERMINALS.
  167 022200     ADD 1 TO COUNTER.
      022300
      022400 ACQUIRE-AND-INVITE-TERMINALS.
```

Figure 33 (Part 8 of 10). Example of the Use of Multiple Device Files

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  168  022500     ACQUIRE LIST-OF-TERMINALS(COUNTER) FOR MULTIPLE-FILE.
  169  022600     WRITE MULTIPLE-REC                              C/xx)
       022700         FORMAT IS "SIGNON"
       022800         TERMINAL IS LIST-OF-TERMINALS(COUNTER).
       022900                                      Draft
       023000 POLL-TERMINALS.
  170  023100     READ MULTIPLE-FILE RECORD. 13
  171  023200     IF MULTIPLE-FS2 EQUAL "310" THEN
  172  023300         SET NO-DATA-AVAILABLE TO TRUE. 14
  173  023400     IF DATA-AVAILABLE THEN
  174  023500         MOVE MULTIPLE-DEVICE-NAME TO CURRENT-TERMINAL
  175  023600 15       PERFORM PASSWORD-VALIDATION.
       023700
       023800 PASSWORD-VALIDATION.
  176  023900     MOVE CURRENT-TERMINAL TO PASSKEY OF PASSWORD-REC.
  177  024000     READ PASSWORD-FILE RECORD.
  178  024100     IF PASSWORD OF SIGNON-I EQUAL PASSWORD OF PASSWORD-REC THEN
  179  024200         CALL "UPDATE" USING CURRENT-TERMINAL
       024300     ELSE
  180  024400         MOVE "INVALID PASSWORD" TO WRONG OF SIGNON-O.
  181  024500     WRITE MULTIPLE-REC
       024600         FORMAT IS "SIGNON"
       024700         TERMINAL IS CURRENT-TERMINAL.
       024800
       024900 DROP-TERMINALS.
  182  025000     DROP LIST-OF-TERMINALS(COUNTER) FROM MULTIPLE-FILE. 16
                        * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 33 (Part 9 of 10). Example of the Use of Multiple Device Files

1 ATTR is the mnemonic-name associated with the function-name ATTRIBUTE-DATA. ATTR will be used in the ACCEPT statement to obtain attribute data for the TRANSACTION file MULTIPLE-FILE. See item **9A** .

2 File MULT must have been created using the CRTMXDF CL command, where the ACQPGMDEV parameter has a value of *NONE and the MAXPGMDEVE parameter has a value greater than 1. The WAITRCD parameter specifies the wait-time for READ operations on the file. The WAITRCD parameter must have a value greater than 0. In addition, the required devices must have been added to the file by the ADDDSPDEVE CL command.

3 MULTIPLE-FS2 is the extended file status for the TRANSACTION file MULTIPLE-FILE. This variable has been declared in the WORKING-STORAGE section of the program. See item **7** .

4 MULTIPLE-CONTROL-AREA is the control area for the TRANSACTION file MULTIPLE-FILE. This variable will be used to determine which program device was signed on to. See item **15** .

5 The data description for MULTIPLE-REC has been defined using the COPY DDS statement. Note that only the fields which are copied are named fields. Refer to the DDS of this example for comments regarding the DDS used.

6 Format SIGNON is the format with the INVITE keyword. This is the format that will be used to invite devices via the WRITE statement.

7 This is the declaration for the extended file-status MULTIPLE-FS2. It is a 4-byte field which is subdivided into a major return code (first two bytes) and a minor return code (last two bytes).

8 STATION-ATTR is the structure which will be used by the ACCEPT statement to hold the attribute data for the TRANSACTION file MULTIPLE-FILE. See item **9A** .

9 In this statement the extended file status MULTIPLE-FS2 is being written.

9A This is an example of accepting attribute-data for the TRANSACTION file MULTIPLE-FILE. Because we are not interested in a specific program device, but rather the last program device used, the FOR phrases are not used with the ACCEPT.

10 This statement opens the TRANSACTION file MULTIPLE-FILE. Since the ACQPGMDEV parameter of the CRTMXDF command has the value *NONE, no program devices are implicitly acquired during this open.

11 This statement acquires the program device contained in the variable LIST-OF-TERMINALS (COUNTER), for the TRANSACTION file MULTIPLE-FILE.

12 This WRITE statement is inviting the program device specified in the TERMINAL phrase. We know it is inviting the program device because the format SIGNON has the DDS keyword INVITE associated with it. Refer to item **13** .

13 This READ statement will read from any invited program device. See item **12** . If the wait time expires before anyone inputs to the invited devices, the extended file status will be set to “0310” and processing will continue. See item **14** .

14 In this statement, the extended file status for MULTIPLE-FILE is being checked to see if the wait-time expired.

15 The program device name stored in the control area is used to determine which program device was signed on to. See item **4** .

16 This DROP statement detaches the program device contained in the variable LIST-OF-TERMINALS from the TRANSACTION file MULTIPLE-FILE.

Figure 33 (Part 10 of 10). Example of the Use of Multiple Device Files

Program Described Transaction Files

Normally, COBOL TRANSACTION files are externally described. However, if these files are program described, only simple display formatting can be processed. All field level descriptions are defined in the COBOL program.

Packed or binary data (COMP, COMP-3, or COMP-4) should not be sent to a display station as output data. Such data can contain display station control characters which can cause unpredictable results.

Environment Division

File-Control Entry

The TRANSACTION file must be named by a file-control entry in the FILE-CONTROL paragraph. This entry also specifies other information related to the file.

Format

```
SELECT file-name
                                ********************************
     ASSIGN TO assignment-name-1 * [ , assignment-name-2 ] ... *
                                ********************************

     ORGANIZATION IS TRANSACTION

   [ ACCESS MODE IS { SEQUENTIAL                          } ]
                    { DYNAMIC, RELATIVE KEY IS data-name-3 }

   [ FILE STATUS IS data-name-1 { , data-name-5 } ]

   [ CONTROL-AREA is data-name-6 ] .
```

ASSIGN Clause

The ASSIGN clause associates the TRANSACTION file with a display file, communications file, BSC file, or mixed file through the use of assignment-name-1.

The following structure pertains to assignment-name-1:

Format

```
device [ – file name [ – attribute ] ]
```

Device specifies the type of device associated with the file. The value must be WORKSTATION.

file name is a 1 to 10 character external name of the display file, communications file, BSC file, or mixed file.

Attribute specifies the file level option for a separate indicator area, SI. See "Indicators" on page 92 earlier in this chapter.

The second and subsequent assignment-names are syntax-checked, but are treated as documentation.

ORGANIZATION Clause

The ORGANIZATION clause specifies the logical structure of a file. TRANSACTION organization signifies interaction between a COBOL program and either a work station user or another system.

TRANSACTION Organization: TRANSACTION processing can be characterized as the random arrival of a record from one of multiple possible sources followed by appropriate processing, and finally, by the output of results or feedback information of some type to the source of the record.

In some cases, all records are homogenous; that is, a logical transaction is completed with one exchange of records. In other situations, a series of records is passed back and forth in a logical progression with various record types either being selected by the initiator or as part of the processing based on input data values.

Each transaction can be processed by a different program, or multiple transactions can be processed by the same program, depending on the system environment.

The initiation of a transaction can cause a program to be scheduled to process the transaction.

A transaction can consist of a series of alternating requests and responses (a dialogue). Each request and response can consist of multiple logical records.

ACCESS MODE Clause

The ACCESS MODE clause and the RELATIVE KEY clause are discussed under "FILE-CONTROL Paragraph" in Chapter 8, "Identification and Environment Divisions"

For files with TRANSACTION organization, the access mode can be SEQUENTIAL or DYNAMIC.

When ACCESS IS SEQUENTIAL is specified or implied, the format name contained in the format name field of the control area specifies which record was accessed. When ACCESS IS SEQUENTIAL is specified for a TRANSACTION file, the RELATIVE KEY data item must not be specified.

When ACCESS IS DYNAMIC is specified, records in the file can be accessed sequentially or randomly, depending on the form of the specific input/output request. Random accessing of a TRANSACTION file is only valid if subfile processing is being processed. For subfile processing, ACCESS IS DYNAMIC *must* be specified.

RELATIVE KEY Clause

The RELATIVE KEY clause specifies the relative record number for a specific record in a subfile. The RELATIVE KEY data item, data-name-3, must be defined as an unsigned integer and must not be defined in a record description entry associated with the TRANSACTION file.

FILE STATUS Clause

General considerations about the FILE STATUS clause and data-name-1 are described under "FILE-CONTROL Paragraph" in Chapter 8, "Identification and Environment Divisions"

Data-name-5 identifies the extended file status data item, which contains major and minor return codes. These major and minor return codes can, in some cases, indicate I-O errors when the file status code does not.

Data-name-5 must be defined in the Data Division as a 4-byte alphanumeric data item, and must *not* be defined in the File Section. The first two bytes of the extended file status data item contain the major return code, and the second two bytes contain the minor return code. Return codes are moved into data-name-5 after any input or output operation (except the ACCEPT or CLOSE statement) on the TRANSACTION file. The values placed in data-name-5 can also be accessed by the ACCEPT statement using the I-O-FEEDBACK function-name.

CONTROL-AREA Clause

The CONTROL-AREA clause specifies device dependent and system dependent information that is used to control input/output operations for TRANSACTION files.

Data-name-6 is a CONTROL-AREA data item that must be defined in the LINKAGE SECTION or WORKING-STORAGE SECTION. Data-name-6 is assumed to have the following format:

```
01  data-name-6.
    02  data-name-12 PIC X(2).
              (Function key feedback field)
    02  data-name-11 PIC X(10).
              (Program device name)
    02  data-name-10 PIC X(10).
              (Record format)
```

The data items associated with data-name-6 must be 2, 12, or 22 characters long. Based upon this length, the compiler assumes the availability of key feedback bytes, the program device name, and record format.

Note: For a mixed file, the actual name of a device may be different than the program device name (data-name-11).

Information is moved into data-name-6 for each READ operation from a file that has been assigned to a WORKSTATION device type. The information is valid only if the READ operation is successfully completed (provided the wait time has not expired).

The information is in the fixed format as shown in the following example:

```
FILE-CONTROL.
SELECT SCREEN-FILE
    ASSIGN TO WORKSTATION-MYFMTS
    ORGANIZATION IS TRANSACTION
    CONTROL-AREA IS
     TRANSACTION-CONTROL-AREA.
    .
    .
    .
WORKING-STORAGE SECTION.
01  TRANSACTION-CONTROL-AREA.
*    FEEDBACK ITEM
    02  COMMON-AREA.
        03  FUNCTION-KEY  PIC XX.
        03  TERMINAL-ID   PIC X(10).
    02  FORMAT-NAME      PIC X(10).
```

Each field in the TRANSACTION-CONTROL-AREA data item in the example is described as follows:

- FUNCTION-KEY: A two-digit number inserted in the field by the work station interface that identifies which function key the operator pressed to initiate the transaction. The codes are as follows:

Code	Function Key
00	Enter key
01-24	Function keys 1 through 24
90	Roll Up key
91	Roll Down key
92	Print key
93	Help key
94	Clear key
95	Home key
99	Undefined

  Any function keys for which feedback information is desired must be defined for the display file or mixed file using DDS. The Print key must also be optioned by a response indicator before feedback information can be provided in the function key field of the CONTROL-AREA data-name.

- TERMINAL-ID: The *program device name.*
- FORMAT-NAME: The DDS record format name that was referenced by the last I-O statement processed.

Data Division

File Description Entry

A file description entry consists of a level indicator (FD), a file-name, and a series of independent clauses. For a TRANSACTION file, the independent clauses allowed are the RECORD CONTAINS clause, the LABEL RECORDS clause, and the DATA RECORDS clause. Only the LABEL RECORDS clause is required.

Format

```
[ FD file-name

  [ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]

  ************************************
  * LABEL { RECORD IS   } { OMITTED  } *
  *       { RECORDS ARE } { STANDARD } *
  *                                    *
  ************************************

  [ DATA { RECORD IS   } data-name-3 [ , data-name-4 ] . . . ] .
         { RECORDS ARE }

{ record-description-entry } . . . ]
```

The LABEL RECORDS clause specifies whether or not labels are present. This clause is required in every file description entry. This clause is syntax-checked, but is treated as documentation.

The RECORD CONTAINS clause and the DATA RECORDS clause are described in Chapter 9, “Data Division.”

Boolean Data Facilities

The use of Boolean data and the use of indicators is described under “Indicators” on page 92.

Procedure Division

See Appendix E, “File Structure Support Summary and Status Key Values” for a file structure support summary.

ACCEPT Statement

The ACCEPT statement retrieves information (attribute data) about a particular program device associated with a TRANSACTION file.

Format 5–TRANSACTION Attributes

```
ACCEPT identifier-1  FROM mnemonic-name

[ FOR { identifier-2 } [ FOR file-name ] ] .
      { literal      }
```

This format of the ACCEPT statement may only be used for files with an organization of TRANSACTION. If the file is not open at the time the ACCEPT statement is proc-

essed, processing stops and message CBE7205 is issued and the program stops running. Mnemonic-name must be associated with the function-name ATTRIBUTE-DATA in the SPECIAL-NAMES paragraph.

If file-name is not specified, the default file for the ACCEPT statement is the first TRANSACTION file specified in a SELECT clause of the FILE-CONTROL paragraph.

Literal or the contents of identifier-2, if specified, indicates the program device name for which attribute data is made available. This device must have been defined (through a CRTxxxF, CHGxxxF, OVRxxxF, or ADDxxxDEVE CL command, where xxx has a value of BSC, CMN, or DSP) as available to be acquired by the file, but need not have actually been acquired. Literal, if specified, must be non-numeric and 10 characters or less in length. The contents of identifier-2, if specified, must be an alphanumeric data item 10 characters or less in length. If an invalid program device name is specified, message CBE7205 is issued and the program stops running.

If both FOR phrases are omitted (indicating the default TRANSACTION file is being used) the ACCEPT statement uses the program device from which a READ, WRITE, REWRITE, or ACCEPT (Attribute Data) operation on the default file was most recently processed. If the only prior operation on the file was an OPEN, the ACCEPT statement uses the program device implicitly acquired by the file when the file was opened. When both FOR phrases are omitted, a program device must have been acquired in order to use this format of the ACCEPT statement.

Program device attributes are moved into identifier-1 from the appropriate attribute data format, according to the rules for a group MOVE without the CORRESPONDING phrase.

Attribute Data Formats

The attribute data retrieved by the ACCEPT statement has two different formats, depending on whether the data is for a work station or for a communications device. See Appendix E, “File Structure Support Summary and Status Key Values” for format descriptions.

Note: For a mixed file containing different types of program devices, you may need to test the attribute data to determine the type of program device for which information has been returned. This is done by testing the first byte of the attribute data, which has different values for work stations and communications devices.

The ATTRIBUTE-DATA mnemonic name can be used *only* to obtain information about a program device acquired by a TRANSACTION file. Attribute data does *not* provide information about the status of a completed or attempted I-O operation. To obtain information about I-O operations, use the Format 3 ACCEPT statement with the I-O-FEEDBACK or OPEN-FEEDBACK mnemonic names. For more information about these mnemonic names, see “ACCEPT Statement” on page 373.

ACQUIRE Statement

The ACQUIRE statement acquires a program device for a TRANSACTION file.

Format

```
ACQUIRE { identifier } FOR file-name
        { literal    }
```

Literal or the contents of identifier indicates the program device name to be acquired by the specified file. Literal, if specified, must be non-numeric and 10 characters or less in length. Identifier, if specified, must refer to an alphanumeric data item 10 characters or less in length.

File-name must be the name of a file with an organization of TRANSACTION, and the file must be open when the ACQUIRE statement is processed. A compilation error message is issued if the organization is not TRANSACTION.

Successful completion of the ACQUIRE operation makes the program device available for input and output operations. If the ACQUIRE is unsuccessful, the file status value is set to 9H and any applicable USE AFTER EXCEPTION/ERROR procedure is called.

Only one program device may be implicitly acquired when a file is opened. If a file is a mixed file, the single implicitly acquired program device is determined by the ACQPGMDEV parameter of the CRTMXDF CL command. If the file is a display file, the single implicitly acquired program device is determined by the first entry in the DEV parameter of the CRTDSPF, CHGDSPF, or OVRDSPF CL command. Additional program devices *must* be explicitly acquired. If the file is a communications or BSC file, the single implicitly acquired device is determined by the DEV parameter of the CRTCMNF, CRTBSCF, CHGCMNF, CHGBSCF, OVRCMNF, or OVRBSCF CL command. Communications and BSC files can never acquire multiple program devices.

A program device is explicitly acquired by using the ACQUIRE statement. For a mixed file, that device must have been added to the file with an ADDDSPDEVE, ADDCMNDEVE or ADDBSCDEVE CL command before the file was opened. For a display file, the program device name must be the same as the display device name.

The ACQUIRE statement can also be used as an aid in recovering from I-O errors. For more information, see “Transaction File Recovery” on page 257.

CLOSE Statement

The CLOSE statement terminates the processing of files.

Format 2

```
CLOSE file-name-1 [ WITH LOCK ]
  [ file-name-2 [ WITH LOCK ] ] . . .
```

For a further discussion of the CLOSE statement, see “CLOSE Statement” on page 377.

DROP Statement

The DROP statement releases a program device that has been acquired by a TRANSACTION file.

Format

```
DROP { identifier }  FROM file-name
     { literal    }
```

Literal or the contents of identifier indicates the program device name of the device to be dropped. Literal, if specified, must be non-numeric and 10 characters or less in length. Identifier, if specified, must refer to an alphanumeric data item, 10 characters or less in length.

File-name must refer to a file with an organization of TRANSACTION, and the file must be open in order to be used in the DROP statement. If no DROP statement is issued, program devices attached to a TRANSACTION file are implicitly released when that file is finally closed.

Program devices specified in a DROP statement must have been acquired by the TRANSACTION file, either through an explicit ACQUIRE or through an implicit ACQUIRE at OPEN time.

After successful processing of the DROP statement, the program device is no longer available for input or output operations through the TRANSACTION file. The device may be reacquired if necessary. The contents of the record area associated with a released program device are no longer available, even if the device is reacquired.

If the DROP statement is unsuccessful, any applicable USE AFTER EXCEPTION/ERROR procedures are run.

The DROP statement can also be used as an aid in recovering from I-O errors. For more information, see “Transaction File Recovery” on page 257.

OPEN Statement

The OPEN statement initiates the processing of files.

Format 3–TRANSACTION Files

```
OPEN I-O file-name-1 [ file-name-2 ] . . .
```

A TRANSACTION file must be opened in the I-O mode. For a further discussion of the OPEN statement, see “OPEN Statement” on page 389.

The OPEN statement can cause a program device to be implicitly acquired for a TRANSACTION file. For a further discussion about the acquiring of program devices, see “ACQUIRE Statement” on page 127.

Common Processing Facilities

The following discussion on FORMAT, INDICATORS, SUBFILE, and TERMINAL phrases relates to the READ, REWRITE, and WRITE statements.

FORMAT Phrase

The literal or identifier specified must be a character-string of 10 characters or less in length.

Multiple data records, each with a different format, can be concurrently active for a TRANSACTION file. If the FORMAT phrase is specified, it must specify a valid format name that is defined to the system, and the I-O operation must be processed on a data record of the same format. If the format is an invalid name or if it does not exist, the FILE STATUS data item, if specified, is set to a value of 9K and the contents of the record area are undefined.

DB-FORMAT-NAME Special Register

After the processing of an input/output statement for a TRANSACTION file, the DB-FORMAT-NAME special register is modified according to the following rules:

- If the input/output operation is successful, the record format name is implicitly moved to the special register after completion of the input/output operation.
- If the input/output operation is unsuccessful, DB-FORMAT-NAME contains the record format name used in the last successful input/output operation.

When the FORMAT phrase is not specified, DB-FORMAT-NAME can be used if the file contains a default record format name. The default value is always moved to the DB-FORMAT-NAME special register.

DB-FORMAT-NAME is implicitly defined as PICTURE X(10).

INDICATORS Phrase

The identifier specified in the INDICATORS phrase must be either an elementary Boolean data item specified without the OCCURS clause or a group item that has elementary Boolean data items subordinate to it.

When a data record is written or rewritten, indicators can be written or rewritten with it. The indicators can control how the record is displayed and also the various Data Management functions.

When a data record is read, indicators can be read with it. The indicators can be used to pass information about the data record and how it was entered into the user program.

The user determines, when he defines a format using DDS, what functions are to be controlled by indicators, and which indicator(s) controls a particular function.

See “Indicators” on page 92 for more information on the INDICATORS phrase.

SUBFILE Phrase

When the SUBFILE phrase is specified, it indicates that all formats referenced by the statement are subfiles. When SUBFILE is not specified in a TRANSACTION I-O statement, it indicates that none of the formats referenced by the statement are subfiles. This information is not verified at compile-time. If it is specified incorrectly, an incorrect program is generated; when the program is run, the FILE STATUS data item, if specified, is set to a value of 92 (logic error), and the contents of the record are undefined.

When SUBFILE is not specified, the RELATIVE KEY data item associated with the file, if specified, is not referenced or changed by the I-O operation.

When SUBFILE is specified, a RELATIVE KEY data item must be defined for the file. Its value is referenced, and sometimes changed, by the I-O operation. See each I-O statement associated with SUBFILE operations for a detailed description of when and how the RELATIVE KEY data item is changed.

The SUBFILE phrase can be specified only for display files, and for display devices in a mixed file.

TERMINAL Phrase

When the TERMINAL phrase is specified, it indicates a specific program device is to be used for a READ, WRITE, or REWRITE operation on a TRANSACTION file.

The TERMINAL phrase can be omitted for I-O operations on single device files, since the single device is always used.

If the TERMINAL phrase is omitted for an I-O operation on a TRANSACTION file that has acquired multiple program devices, the program device that last attempted a READ, WRITE, REWRITE, ACQUIRE, DROP, or ACCEPT (Attribute Data) operation on the file is used. If the only prior operation on the file was an OPEN, the default program device used is the program device implicitly acquired by the TRANSACTION file when the file was opened. A run-time error message occurs if no program device is acquired when the file is opened.

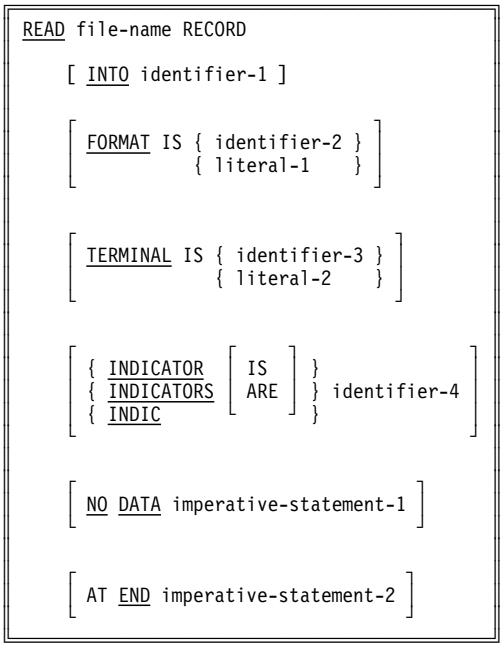
For a READ statement with both the TERMINAL phrase and the NO DATA phrase specified, the imperative-statement in the NO DATA phrase is processed only if data is not immediately available from the program device specified by the TERMINAL phrase.

If the TERMINAL phrase is specified and the data-item or literal has a value of blanks, the phrase is treated at run time as if it were not specified.

READ Statement

The READ statement makes available a record from a device, using a named format. If the format is a subfile, the READ statement makes available a specified record from that subfile.

Format 4–TRANSACTION File (Nonsubfile)



Format 4 is used only to read a format that is not a subfile. The RELATIVE KEY data item, if specified in the FILE-CONTROL entry, is not used. The Format 4 READ statement is not valid for a subfile record. However, a Format 4 READ statement for the subfile control record format must be used to place those subfile records that were updated on a display into the subfile.

If the data is available, it is returned in the record area. The names of the record format and the program device are returned in the I-O-FEEDBACK area in the CONTROL-AREA.

The READ statement is valid only when there are acquired devices for the file. If a READ is processed and there are no acquired devices, the file status is set to 92 (logic error).

The manner in which the Format 4 READ statement functions depends on:

- If the READ is for a single device file or a multiple device file
- If a specific program device has been requested through the TERMINAL phrase
- If a specific record format has been requested through the FORMAT phrase
- If the NO DATA phrase has been specified.

In the following discussions, references to “data available or returned” include the situation where only the response indicators are set. This is so even when a separate indicator area is used and the indicators are not returned in the record area for the file.

The following chart shows the possible combinations of phrases, and the function processed for a single device file or a multiple device file. For example, if TERMINAL

is N, FORMAT is N, and NO DATA is N, then the single device is D and multiple device is A.

	Phrase	Y=Yes N=No
Checked at Compilation	TERMINAL[4] FORMAT[4] NO DATA	N N N N Y Y Y Y N N Y Y N N Y Y N Y N Y N Y N Y
Determined at Run Time	Single Device Multiple Device	D C D B D C D B A A D B D C D B

Codes A through D are explained below.

Code A–Read From Invited Program Device (Multiple Device Files only)

This type of READ receives data from the first invited program device that has data available. An invited program device is a work station or communications device (LU1, BSC, or APPC) that has been invited to send input. Inviting is done by writing to the program device with a format that has the DDS keyword INVITE specified. Once an invited program device is actually read from, it is no longer invited. That program device will not be used for input by another READ statement unless reinvited, or unless a READ is directed to it specifying the TERMINAL phrase or FORMAT phrase.

The record format returned from the program device is determined by the system.

This READ can complete without returning any data in the following cases:

1. There are no invited devices. This is the AT END condition, which occurs when:
   - There are no invited devices.
   - For an APPC device, another READ is done after a detach signal is received.
   - For an LU1 device, the session is terminated by the host.
2. A controlled cancel of the job occurs. This results in a file status value of 9A and a major-minor return code value of 0309.
3. The NO DATA phrase is omitted and the specified wait time expires. This results in a file status value of 00 and a major-minor return code value of 0310. The specified wait time is the value entered on the WAITRCD parameter for the file.
4. The NO DATA phrase is specified and there is no data immediately available when the READ is processed.

If data is available, it is returned in the record area. The record format is returned in the I-O-FEEDBACK area and in the CONTROL-AREA.

Code B–Read From One Program Device (Invalid combination)

A compilation time message is issued and the NO DATA phrase is ignored. See the table entry for the same combination of phrases with the NO DATA phrase omitted.

[4] If the phrase is specified and the data item or literal is blank, the phrase is treated at run time as if it were not specified.

Code C–Read From One Program Device (with NO DATA phrase)

This function of the READ statement never causes program processing to stop and wait until data is available. Either the data is immediately available or the NO DATA imperative-statement is processed.

This READ function can be used to periodically check if data is available from a particular program device (either the default program device or one specified by the TERMINAL phrase). This checking for data is done in the following manner:

1. The program device is determined as follows:
   a. If the TERMINAL phrase was omitted or contains blanks, the default program device is used. The default program device is the one used by the last attempted READ, WRITE, REWRITE, ACQUIRE, or DROP statement. If none of the above I-O operations were previously processed, the default program device is the first program device acquired.
   b. If the TERMINAL phrase was specified, the indicated program device is used.
2. A check is done to determine if data is available and if the program device is invited.
3. If data is available, that data is returned in the record area and the program device is no longer invited. If no data is immediately available, the NO DATA imperative-statement is processed and the program device remains invited.
4. If the program device is not invited, the AT END condition exists and the file status is set to 10.

Code D–Read From One Program Device (without NO DATA Phrase)

This READ always waits for data to be made available. Even if the job receives a controlled cancel, or a WAITRCD time is specified for the file, the program will never regain control from the READ statement. This READ operation is processed in the following manner:

1. The program device is determined as follows:
   a. If the TERMINAL phrase is omitted or contains a blank value, the default program device is used. The default program device is the program device used by the last attempted READ, WRITE, REWRITE, ACQUIRE, DROP or ACCEPT (Attribute Data) statement. If none of these operations has been done, the program device implicitly acquired when the file was opened is used. If there are no acquired devices, the AT END condition exists.
   b. If the TERMINAL phrase is specified, the indicated program device is used.
2. The record format is determined as follows:
   a. If the FORMAT phrase is omitted or contains blanks, the record format returned is determined by the system.
   b. If the FORMAT phrase is specified, the indicated record format is returned. If the data available does not match the requested record format, a file status of 9K is set.
3. Program stops running until data becomes available. The data is returned in the record area after the READ statement is processed. If the program device was previously invited, it will no longer be invited after this READ statement.

4. The AT END condition can only occur for LU1 or APPC devices. When there are multiple LU1 or APPC devices acquired for a TRANSACTION file, each individual device can cause an AT END condition to occur. The AT END condition occurs when:
   - For an APPC device, another READ is done after a detach signal is received. Since a detach signal can be sent with or without data, check the major-minor return codes to determine if there was any data.
   - For an LU1 device, the session is terminated by the host.

AT END Phrase

When the AT END condition is detected, imperative-statement-2 is processed.

FORMAT Phrase

The name of the record format to be read is specified by literal-1 or identifier-2. If literal-1 is specified, it must be nonnumeric and 10 characters or less in length. If identifier-2, is specified, must refer to an alphanumeric data item, 10 characters or less in length. If identifier-2 contains blanks, the READ statement is processed as if the FORMAT phrase was omitted.

NO DATA Phrase

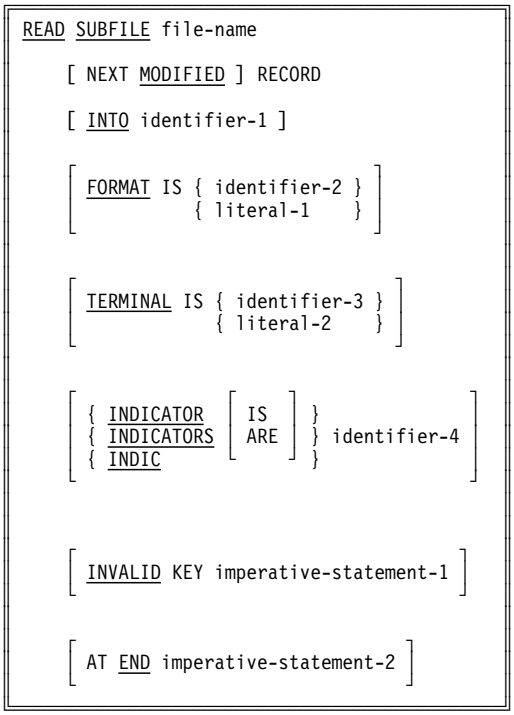
When the NO DATA phrase is specified, the READ statement will determine whether data is immediately available. If data is available, the data is returned in the record area. If no data is immediately available, imperative-statement-1 is processed. The NO DATA phrase prevents the READ statement from waiting for data to become available.

TERMINAL Phrase

The program device name is specified by literal-2 or identifier-3. If literal-2 is specified, must be nonnumeric and 10 characters or less in length. If identifier-3 is specified, must refer to an alphanumeric data item, 10 characters or less in length. The program device must have been acquired before the READ statement is processed. If identifier-3 contains blanks, the READ statement is processed as if the TERMINAL phrase was omitted. For a single device file, the TERMINAL phrase can be omitted. The program device is assumed to be that single device.

If the TERMINAL phrase is omitted for a READ of a TRANSACTION file that has acquired multiple program devices, the default program device is used. See the general discussion about the TERMINAL phrase in the “Common Processing Facilities” section earlier in this chapter for details about how the default program device is determined.

Format 5–TRANSACTION File (Subfile)



Format 5 is used only to read a format that is a subfile. The AT END phrase can only be used when the NEXT MODIFIED phrase is specified. The INVALID KEY phrase must not be used when the NEXT MODIFIED phrase is specified.

Format 5 cannot be used for communications devices. If the subfile format of the READ statement is used for a communications device, the READ fails and a file status of 90 is set.

Random Access of Subfile Records: The NEXT MODIFIED phrase must not be used to randomly access records in a subfile. The INVALID KEY phrase can only be used for random access of subfile records.

Sequential Access of Subfile Records: The NEXT MODIFIED phrase must be specified to access subfile records sequentially. The AT END phrase can only be specified with the NEXT MODIFIED phrase.

NEXT MODIFIED Phrase

When NEXT MODIFIED is not specified, the data record made available is the record in the subfile with a relative record number that corresponds to the value of the RELATIVE KEY data item.

When the NEXT MODIFIED phrase is not specified, and if the RELATIVE KEY data item contains a value other than the relative record number of a record in the subfile, the INVALID KEY condition exists and the processing of the READ statement is unsuccessful.

When the NEXT MODIFIED phrase is specified, the record made available is the first record in the subfile that has been modified (has the Modified Data Tag on).

The search for the next modified record begins:

- At the beginning of the subfile if:
  - An I-O operation has been processed for the subfile control record.
  - The I-O operation cleared, initialized, or displayed the subfile.
- For all other cases, with the record following the record that was read by a previous read operation.

The value of the RELATIVE KEY data item is updated to reflect the relative record number of the record made available to the program.

If NEXT MODIFIED is specified and there is no user-modified record in the subfile with a relative record number greater than the relative record number contained in the RELATIVE KEY data item, the AT END condition exists. Imperative-statement-2, or any applicable USE AFTER ERROR/EXCEPTION procedure, if any, is then run.

FORMAT Phrase

When a format-name is not specified, the format used is last record format written to the display device that contains input fields, input/output fields, or hidden fields. If no such format exists for the display file, the format used is the record format of the last WRITE operation to the display device.

If the FORMAT phrase is specified, literal-1 or the contents of identifier-2 must specify a format, which is active for the appropriate program device. The READ statement reads a data record of the specified format.

The FORMAT phrase should always be specified for multiple format files to ensure correct results. For more information on the FORMAT phrase, see “Common Processing Facilities” on page 130.

TERMINAL Phrase

See Format 4 above for general considerations concerning the TERMINAL phrase.

For a Format 5 READ, if the TERMINAL phrase is omitted for a file that has multiple devices acquired for it, a record is read from the subfile associated with the default program device. See the general discussion about the TERMINAL phrase in the “Common Processing Facilities” on page 130 for more details about how the default program device is determined.

INVALID KEY Phrase

If the RELATIVE KEY data item at the time of the processing of the READ statement contains a value that does not correspond to a relative record number for the subfile, the INVALID KEY condition exists and the processing of the READ statement is unsuccessful.

For a Format 5 READ, the INVALID KEY phrase should be specified if the NEXT MODIFIED phrase is not specified and there is no applicable USE procedure specified for the file-name.

If both an INVALID KEY phrase and a USE procedure are specified for the file when the INVALID KEY condition occurs, control transfers to the INVALID KEY imperative-statement, and the USE procedure is not run.

AT END Phrase

If NEXT MODIFIED is specified and there is no user-modified record in the subfile, the AT END condition exists, and the processing of the READ statement is unsuccessful.

The AT END phrase should be specified when the NEXT MODIFIED phrase is used, and no applicable USE procedure is specified for the file-name. If the AT END phrase and a USE procedure are both specified for a file, and the AT END condition arises, control transfers to the AT END imperative statement and the USE procedure is not run.

For a further discussion of the READ statement (and a related topic, the INTO phrase), the INVALID KEY phrase, and the AT END phrase, see “READ Statement” on page 393.

REWRITE Statement

The REWRITE statement is used to replace a subfile record that already exists in the subfile.

Format 2–TRANSACTION

```
REWRITE SUBFILE record-name [ FROM identifier-1 ]

     FORMAT IS { identifier-2 }
               { literal-1    }

     [ TERMINAL IS { identifier-3 } ]
                   { literal-2    }

     [ { INDICATOR  [ IS  ] }                    ]
     [ { INDICATORS [ ARE ] } identifier-4       ]
     [ { INDIC              }                    ]

     [ INVALID KEY imperative-statement ]
```

The number of character positions in the record referenced by record-name must be equal to the number of character positions in the record being replaced. A successful read operation on the record must be done prior to the rewrite operation. The record replaced in the subfile is the record in the subfile accessed by the previous read operation.

FORMAT Phrase

The record format specified in the FORMAT phrase must be the record format accessed on the previous read operation. The contents of identifier-2 or literal-1 must be the name of the subfile format accessed on the previous READ. For more information on the FORMAT phrase, see “Common Processing Facilities” on page 130.

TERMINAL Phrase

The TERMINAL phrase indicates which program device's subfile is to have a record rewritten. If the TERMINAL phrase is specified, literal-2 or identifier-3 must refer to a work station that has been acquired by the TRANSACTION file. If literal-2 or identifier-3 contains blanks, the TERMINAL phrase has no effect. The program device specified by the TERMINAL phrase must have been acquired, either explicitly or implicitly, and must have a subfile associated with the device.

Literal-2 or identifier-3 must be a valid program device name. Literal-2, if specified, must be nonnumeric and 10 characters or less. Identifier-3, if specified, must refer to an alphanumeric data item, 10 characters or less in length.

If the TERMINAL phrase is omitted from a TRANSACTION file that has acquired multiple program devices, the subfile used is the subfile associated with the last program device from which a READ of the TRANSACTION file was attempted.

The REWRITE statement cannot be used for communications devices. If the REWRITE statement is used for a communications device, the operation fails and a file status of 90 is set.

INVALID KEY Phrase

If, at the time of the rewrite operation, the RELATIVE KEY data item contains a value that does not correspond to the relative record number of the record from the previous read operation, the INVALID KEY condition exists.

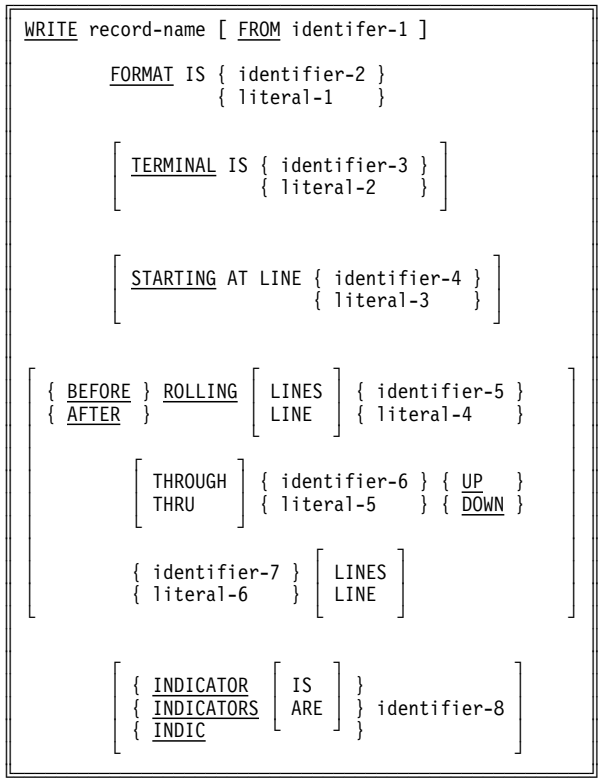
The INVALID KEY phrase should be specified for files for which an appropriate USE procedure is not specified. Undesirable results may occur if the INVALID KEY phrase is not specified, and no USE procedure is specified.

For a further discussion of the REWRITE statement (and the related topic, the FROM phrase) and the INVALID KEY phrase, see "REWRITE Statement" on page 402.

WRITE Statement

The WRITE statement releases a logical record to the file.

Format 4–TRANSACTION File (Nonsubfile)



TERMINAL Phrase

The TERMINAL phrase specifies the program devices to which the output record is to be sent.

The contents of literal-2 or identifier-3 must be the name of a program device previously acquired, either implicitly or explicitly, by the file. Literal-2, if specified, must be nonnumeric and 10 characters or less in length. Identifier-3, if specified, must refer to an alphanumeric data item, 10 characters or less in length. A value of blanks is treated as if the TERMINAL phrase was omitted.

If only a single program device was acquired by the TRANSACTION file, the TERMINAL phrase can be omitted. That program device is always used for the WRITE.

If the TERMINAL phrase is omitted for a WRITE operation to a TRANSACTION file that has acquired multiple program devices, the default program device is used. See the general discussion about the TERMINAL phrase in the “Common Processing Facilities” section earlier in this chapter for details about how the default program device is determined.

STARTING Phrase

The STARTING phrase specifies the starting line number for the record formats that use the variable start line keyword. This phrase is only valid for display devices.

The actual line number on which a field begins can be determined from the following equation:

Actual line number = Starting line number specified in the program + The line number specified in positions 39 through 41 of the Data Description Specification form – 1

The write is successful if:

- The result of the above equation is positive and less than or equal to the number of lines on the work station screen.
- The value specified for the STARTING phrase is 0. In this case, a value of 1 is assumed.

The write is unsuccessful and the program terminates if:

- The result of the above equation is greater than the number of lines on the work station screen.
- The value specified for the STARTING phrase is negative.

If the value specified for the STARTING phrase is within the screen area, any fields outside of the screen area are ignored.

Literal-3 of the STARTING phrase must be a numeric literal. Identifier-4 must be an elementary numeric item.

To use the STARTING phrase, the DDS record level keyword SLNO(*VAR) must be specified for the format being written. If the record format does not specify this keyword, the STARTING phrase is ignored at run time.

The DDS keyword CLRL also affects the STARTING phrase. CLRL controls how much of the screen is cleared when the WRITE statement is processed.

ROLLING Phrase

The ROLLING phrase allows you to move lines displayed on the work station screen. All or some of the lines on the screen can be rolled up or down. The lines vacated by the rolled lines are cleared, and can have another screen format written into them. This phrase is only valid for display devices.

ROLLING is specified in the WRITE statement that is writing a new format to the work station screen. You must specify whether the write is before or after the roll, the range of lines you want to roll, how many lines you want to roll these lines, and whether the roll operation is up or down.

After lines are rolled, the fields on these lines retain their DDS display attributes, for example, underlining, but lose their DDS usage attributes, for example, input-

capability. Fields on lines that are written and then rolled (BEFORE ROLLING phrase) also lose their usage attributes.

If any part of a format is rolled, the entire format loses its usage attributes. If more than one format exists, only the rolled formats lose their usage attributes.

When you specify the ROLLING phrase, the following general rules apply:

- The DDS record level keyword ALWROL must be specified for every record format written in a WRITE statement containing the ROLLING phrase.
- Other DDS keywords mutually exclusive with the ALWROL keyword must not be used.
- Either of the DDS keywords, CLRL or OVERLAY, must be specified for a record format that is to be written and rolled to prevent the display screen from being cleared when that record format is written.
- All the identifiers and literals must represent positive integer values.
- The roll starting line number (identifier-5 or literal-4) must not exceed the ending line number (identifier-6 or literal-5).
- The contents of lines that are rolled outside of the window specified by the starting and ending line numbers disappear.

Figure 34 on page 143 shows an example of rolling. An initial screen format, FMT1 is written on the work station screen. The program processes this screen format and is now ready to write the next screen format, FMT2, to the work station screen. Part of FMT1 is rolled down two lines before FMT2 is written to the work station screen.

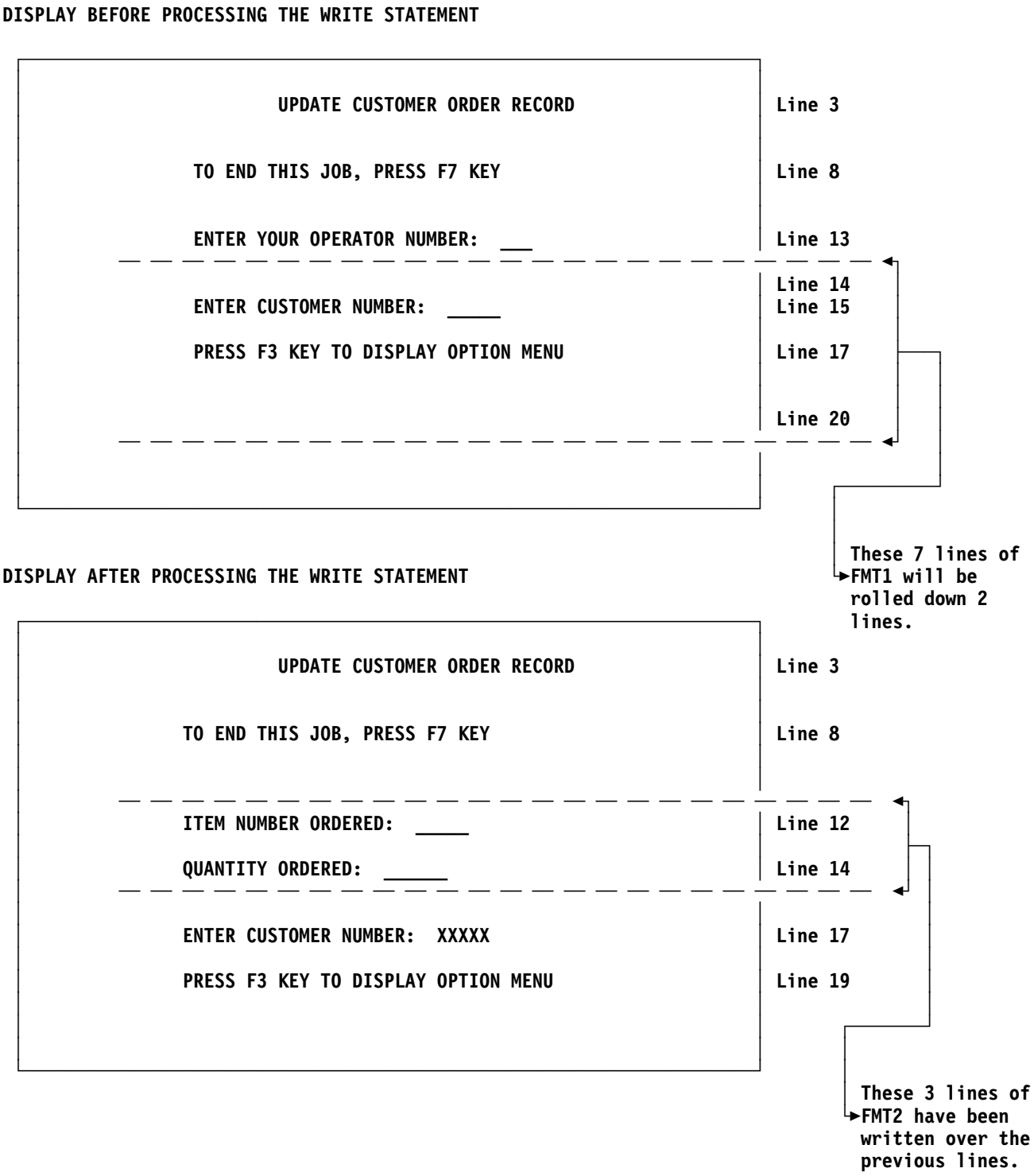


Figure 34. Example of ROLLING Operation

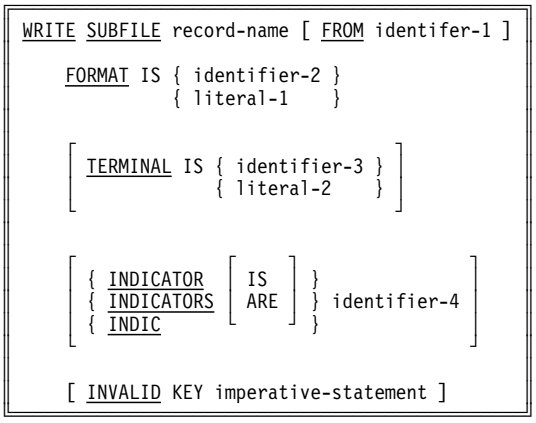
Processing of the following WRITE statement causes part of FMT1 to be rolled down two lines, and FMT2 to be written to the work station screen:

```
WRITE SCREENREC FORMAT "FMT2"
 AFTER ROLLING LINES 14 THROUGH 20
 DOWN 2 LINES
```

When this WRITE statement is processed, the following steps occur:

1. The contents of lines 14 through 20 are rolled down 2 lines.
   a. The contents of lines 14 through 18 now appear on lines 16 through 20.
   b. The contents of lines 14 and 15 are vacated and cleared.
   c. The contents of lines 19 and 20 are rolled outside the window and disappear.
2. After the rolling operation takes place, FMT2 is written to the work station screen.
   a. Part of FMT2 is written to the area vacated by the roll operation.
   b. Part of FMT2 is written over the data left from FMT1.
3. When the contents of the work station screen are returned to the program by a READ statement, only the input capable fields of FMT2 are returned.

Format 5–TRANSACTION File (Subfile)



Format 5 can only be used for display devices. If the subfile form of the WRITE statement is used for any other type of device, the WRITE operation fails and a file status of 90 is set.

If the format is a subfile, and SUBFILE is specified, the RELATIVE KEY clause must have been specified on the SELECT clause for the file being written. The record written to the subfile is the record in the subfile identified by the format name that has a relative record number equal to the value of the RELATIVE KEY data item.

TERMINAL Phrase

See Format 4 above for general considerations concerning the TERMINAL phrase.

The TERMINAL phrase specifies which program device's subfile is to have a record written to it. If the TERMINAL phrase is specified, literal-2 or identifier-3 must refer to a work station associated with the TRANSACTION file. If literal-2 or identifier-3 contains a value of blanks, the TERMINAL phrase is treated as if it was not specified. The work station specified by the TERMINAL phrase must have been acquired, either explicitly or implicitly.

If the TERMINAL phrase is omitted, the subfile used is the subfile associated with the default program device. See the general discussion about the TERMINAL phrase in the "Common Processing Facilities" section earlier in this chapter for details about how the default program device is determined.

The INVALID KEY condition exists if a record is already in the subfile with that record number, or if the relative record number specified is greater than the maximum allowable subfile record number. The INVALID KEY phrase should be specified in the WRITE SUBFILE statement for all files for which an appropriate USE procedure is not specified.

For a further discussion of the WRITE statement, the FROM phrase, and the INVALID KEY phrase, see "WRITE Statement" on page 412. For information on the FORMAT phrase, see "Common Processing Facilities" on page 130.

USE Statement

The USE statement specifies procedures for input/output error handling that are in addition to the standard procedures provided by the input/output control system.

Format

```
USE AFTER STANDARD { ERROR     }
                   { EXCEPTION }

    PROCEDURE ON { file-name-1 [ , file-name-2 ] . . . }
                 { I-O                                 } .
```

See "Declaratives" on page 364 for a further discussion of the USE statement.

Work Station Example Programs

This section contains example COBOL programs that illustrate work station applications on the System/38 environment.

Figure 35 on page 146 shows a basic inquiry program that uses the COBOL TRANSACTION file. The associated DDS for the files is also shown.

The data description specifications (DDS) for the display device file (CUSMINQ) to be used by this program describe two record formats: CUSPMT and CUSFLDS.

The CUSPMT record format contains the constant 'Customer Master Inquiry', which identifies the display. It also contains the prompt 'Customer Number' and the input field (CUST) into which the work station user enters the customer number. Five underscores appear under the input field CUST on the screen where the user is to enter the customer number. The error message 'Customer number not found' is also included in this record format. This message is displayed if indicator 99 is set on by the program. In addition, this record format defines a function key that the user can press to end the program. When the user presses function key 01, indicator 15 is set on in the COBOL program. This indicator is then used to end the program.

The CUSFLDS record format contains the constants 'Name', 'Address', 'City', 'State', 'Zip Code', and 'A/R Balance', which identify the fields to be written out from the

program. This record format also describes the fields that correspond to these constants. All of these fields are described as output fields (blank in position 38) because they are filled in by the program; the user does not enter any data into these fields. To enter another customer number, the user presses the Enter key in response to this record. Notice that the CUSFLDS record is to overlay the CUSPMT record. Therefore, when the CUSFLDS record is written to the screen, the CUSPMT record remains on the screen.

In addition to describing the constants, fields, and attributes for the screen, the record formats also define the line numbers and horizontal positions in which the constants and fields are to be displayed.

Note: The field attributes are defined in a physical file (CUSMSTP) used for field reference purposes, instead of in the DDS for the display file.

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7754-1 UM/060* Printed in U.S.A. *Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic							Description	Page of
Programmer	Date		Key								

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name Not (N)	Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (b/R/I/C/S/O)	Reserved	Name	Reference (R)	Length	Data Type (b/A/P/B/B A/E/X/Y/N/I/1/M)	Decimal Positions	Usage (b/O/I/I B/N/M)	Location: Line	Pos	Functions
	A	*	C U S T O M E R						M		A S T E R I N Q U I R Y F I L E					-	C U S M I N Q		
	A	*																	
	A																		R E F (C U S M S T P)
	A								R		C U S P M T								T E X T (' C U S T O M E R P R O M P T ')
	A																		C A 0 1 (1 5 ' E N D O F P R O G R A M ')
	A																1	3	' C U S T O M E R M A S T E R I N Q U I R Y '
	A																3	3	' C U S T O M E R N U M B E R '
	A										C U S T		R			I	3	2 0	
	A		9 9																E R R M S G (' C U S T O M E R N U M B E R N O T F O U N D +
	A																		P R E S S R E S E T , T H E N E N T E R V A L I D N U M B E +
	A																		R ' 9 9)
	A																5	3	' U S E F 1 T O E N D P R O G R A M , U S E E N T E R
	A																		K E Y T O R E T U R N T O P R O M P T S C R E E N '
	A								R		C U S F L D S								T E X T (' C U S T O M E R D I S P L A Y ')
	A																		C A 0 1 (1 5 ' E N D O F P R O G R A M ')
	A																		O V E R L A Y
	A																8	3	' N A M E '
	A										N A M E		R				8	1 1	
	A																9	3	' A D D R E S S '
	A										A D D R		R				9	1 1	
	A																1 0	3	' C I T Y '
	A										C I T Y		R				1 0	1 1	
	A																1 1	3	' S T A T E '
	A										S T A T E		R				1 1	1 1	
	A																1 1	2 1	' Z I P C O D E '
	A										Z I P		R				1 1	3 1	
	A																1 2	3	' A / R B A L A N C E '
	A										A R B A L		R				1 2	1 7	E D T C D E (J)
	A																		
	A																		

Figure 35 (Part 1 of 6). Example of a TRANSACTION Inquiry Program Using a Single Display Device

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7754-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Not (N)	Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (b/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (b, A/P/S/B, A/S/Y/Y/N/I/M)	Decimal Positions	Usage (b/O/I/B/H/M)	Location: Line	Pos	Functions
	A	*	*	PHYSICAL					R		CUSMST P		CUSTOMER				MASTER		FILE
	A								R		CUSMST								TEXT('CUSTOMER MASTER RECORD')
	A										CUST		5						TEXT('CUSTOMER NUMBER')
	A										NAME		25						TEXT('CUSTOMER NAME')
	A										ADDR		20						TEXT('CUSTOMER ADDRESS')
	A										CITY		20						TEXT('CUSTOMER CITY')
	A										STATE		2						TEXT('STATE')
	A										ZIP		5		00				TEXT('ZIP CODE')
	A										SRHCOD		6						TEXT('CUSTOMER NUMBER SEARCH CODE')
	A										CUSTYP		1		00				TEXT('CUSTOMER TYPE 1=GOV 2-SCH +
	A																		3=BUS 4=PVT 5=OT')
	A										ARBAL		8		02				TEXT('ACCOUNTS REC BALANCE')
	A										ORDBAL		8		02				TEXT('A/R AMT IN ORDER FILE')
	A										LSTAMT		8		02				TEXT('LAST AMOUNT PAID IN A/R')
	A										LSTDAT		6		02				TEXT('LAST DATE PAID IN A/R')
	A										CRDLMT		8		02				TEXT('CUSTOMER CREDIT LIMIT')
	A										SLSYR		10		02				TEXT('CUSTOMER SALES THIS YEAR')
	A										SLSLYR		10		02				TEXT('CUSTOMER SALES LAST YEAR')
	A								K		CUST								

The data description specifications (DDS) for the data base file that is used by this program describe one record format: `CUSMST`. Each field in the record format is described, and the `CUST` field is identified as the key field for the record format.

Figure 35 (Part 2 of 6). Example of a TRANSACTION Inquiry Program Using a Single Display Device

```
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
  100        IDENTIFICATION DIVISION.
  200        PROGRAM-ID.     EXMPLE766.
  300       *    EXAMPLE TRANSACTION INQUIRY PROGRAM USING 1 DISPLAY DEVICE
  400        AUTHOR.       PROGRAMMER NAME.
  500        INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
  600        DATE-WRITTEN. 08/30/88.
  700        DATE-COMPILED.  08/30/88 15:36:14   .
  800        ENVIRONMENT DIVISION.
  900        CONFIGURATION SECTION.
 1000        SOURCE-COMPUTER. IBM-S38.
 1100        OBJECT-COMPUTER. IBM-S38.
 1200        INPUT-OUTPUT SECTION.
 1300        FILE-CONTROL.
 1400            SELECT CUST-DISPLAY
 1500                ASSIGN TO WORKSTATION-CUSMINQ
 1600                ORGANIZATION IS TRANSACTION
 1700                CONTROL-AREA IS WS-CONTROL.
 1800            SELECT CUST-MASTER
 1900                ASSIGN TO DATABASE-CUSMSTP
 2000                ORGANIZATION IS INDEXED
 2100                ACCESS IS RANDOM
 2200                    RECORD KEY IS CUST OF CUSMST
 2300                FILE STATUS IS CM-STATUS.
 2400        DATA DIVISION.
 2500        FILE SECTION.
 2600        FD  CUST-DISPLAY
 2700            LABEL RECORDS ARE OMITTED.
 2800        01  DISP-REC.
 2900            COPY DDS-ALL-FORMATS OF CUSMINQ.
 3000
 3100        FD  CUST-MASTER
 3200            LABEL RECORDS ARE STANDARD.
 3300        01  CUST-REC.
 3400            COPY DDS-CUSMST OF CUSMSTP.
 3500
 3600        WORKING-STORAGE SECTION.
 3700        01  ONE                                 PIC 1 VALUE B"1".
 3800        01  CM-STATUS                           PIC X(2).
 3900        01  WS-CONTROL.
 4000            02  WS-IND                          PIC X(2).
 4100            02  WS-FORMAT                       PIC X(10).
```

The SEU listing of the Identification, Environment, and Data Division statements for this example program is shown here. In particular, note the FILE-CONTROL and FD entries.

Figure 35 (Part 3 of 6). Example of a TRANSACTION Inquiry Program Using a Single Display Device

```
5763CB1                              COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.     EXMPLE766.
       000300*    EXAMPLE TRANSACTION INQUIRY PROGRAM USING 1 DISPLAY DEVICE
    3  000400 AUTHOR.       PROGRAMMER NAME.
    4  000500 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
    5  000600 DATE-WRITTEN. 08/30/88.
    6  000700 DATE-COMPILED.  08/30/88 15:36:14   .
    7  000800 ENVIRONMENT DIVISION.
    8  000900 CONFIGURATION SECTION.
    9  001000 SOURCE-COMPUTER. IBM-S38.
   10  001100 OBJECT-COMPUTER. IBM-S38.
   11  001200 INPUT-OUTPUT SECTION.
   12  001300 FILE-CONTROL.
   13  001400     SELECT CUST-DISPLAY
   14  001500         ASSIGN TO WORKSTATION-CUSMINQ
   15  001600         ORGANIZATION IS TRANSACTION
   16  001700         CONTROL-AREA IS WS-CONTROL.
   17  001800     SELECT CUST-MASTER
   18  001900         ASSIGN TO DATABASE-CUSMSTP
   19  002000         ORGANIZATION IS INDEXED
   20  002100         ACCESS IS RANDOM
   21  002200             RECORD KEY IS CUST OF CUSMST
   22  002300         FILE STATUS IS CM-STATUS.
   23  002400 DATA DIVISION.
   24  002500 FILE SECTION.
   25  002600 FD  CUST-DISPLAY
   26  002700     LABEL RECORDS ARE OMITTED.
   27  002800 01  DISP-REC.
   28  002900     COPY DDS-ALL-FORMATS OF CUSMINQ.
   29 +000001       05  CUSMINQ-RECORD PIC X(80).                                  <-ALL-FMTS
      +000002*  INPUT FORMAT:CUSPMT     FROM FILE CUSMINQ    OF LIBRARY COB38EX     <-ALL-FMTS
      +000003*                          CUSTOMER PROMPT                             <-ALL-FMTS
   30 +000004       05  CUSPMT-I      REDEFINES CUSMINQ-RECORD.                   <-ALL-FMTS
   31 +000005           06 CUSPMT-I-INDIC.                                         <-ALL-FMTS
   32 +000006                07 IN15         PIC 1  INDIC 15.                     <-ALL-FMTS
      +000007*                          END OF PROGRAM                              <-ALL-FMTS
   33 +000008                07 IN99         PIC 1  INDIC 99.                     <-ALL-FMTS
      +000009*                          CUSTOMER NUMBER NOT FOUND PRESS RESET, THE  <-ALL-FMTS
   34 +000010           06 CUST              PIC X(5).                            <-ALL-FMTS
      +000011*                          CUSTOMER NUMBER                             <-ALL-FMTS
      +000012* OUTPUT FORMAT:CUSPMT     FROM FILE CUSMINQ    OF LIBRARY COB38EX     <-ALL-FMTS
      +000013*                          CUSTOMER PROMPT                             <-ALL-FMTS
   35 +000014       05  CUSPMT-O      REDEFINES CUSMINQ-RECORD.                   <-ALL-FMTS
   36 +000015           06 CUSPMT-O-INDIC.                                         <-ALL-FMTS
   37 +000016                07 IN99         PIC 1  INDIC 99.                     <-ALL-FMTS
      +000017*                          CUSTOMER NUMBER NOT FOUND PRESS RESET, THE  <-ALL-FMTS
      +000018*  INPUT FORMAT:CUSFLDS    FROM FILE CUSMINQ    OF LIBRARY COB38EX     <-ALL-FMTS
      +000019*                          CUSTOMER DISPLAY                            <-ALL-FMTS
   38 +000020       05  CUSFLDS-I     REDEFINES CUSMINQ-RECORD.                   <-ALL-FMTS
   39 +000021           06 CUSFLDS-I-INDIC.                                        <-ALL-FMTS
   40 +000022                07 IN15         PIC 1  INDIC 15.                     <-ALL-FMTS
      +000023*                          END OF PROGRAM                              <-ALL-FMTS
      +000024* OUTPUT FORMAT:CUSFLDS    FROM FILE CUSMINQ    OF LIBRARY COB38EX     <-ALL-FMTS
      +000025*                          CUSTOMER DISPLAY                            <-ALL-FMTS
   41 +000026       05  CUSFLDS-O     REDEFINES CUSMINQ-RECORD.                   <-ALL-FMTS
```

The Data Division for this example program is shown after compilation to illustrate the data structures generated by the COPY statements, DDS formats.

Figure 35 (Part 4 of 6). Example of a TRANSACTION Inquiry Program Using a Single Display Device

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   42 +000027           06 NAME              PIC X(25).                                    <-ALL-FMTS
      +000028*                              CUSTOMER NAME                                 <-ALL-FMTS
   43 +000029           06 ADDR              PIC X(20).                                    <-ALL-FMTS
      +000030*                              CUSTOMER ADDRESS                              <-ALL-FMTS
   44 +000031           06 CITY              PIC X(20).                                    <-ALL-FMTS
      +000032*                              CUSTOMER CITY                                 <-ALL-FMTS
   45 +000033           06 STATE             PIC X(2).                                     <-ALL-FMTS
      +000034*                              STATE                                         <-ALL-FMTS
   46 +000035           06 ZIP               PIC S9(5).                                    <-ALL-FMTS
      +000036*                              ZIP CODE                                      <-ALL-FMTS
   47 +000037           06 ARBAL             PIC S9(6)V9(2).                               <-ALL-FMTS
      +000038*                              ACCOUNTS REC. BALANCE                         <-ALL-FMTS
   48  003000
   49  003100 FD  CUST-MASTER
   50  003200     LABEL RECORDS ARE STANDARD.
   51  003300 01  CUST-REC.
   52  003400     COPY DDS-CUSMST OF CUSMSTP.
      +000001*    I-O FORMAT:CUSMST     FROM FILE CUSMSTP    OF LIBRARY COB38EX           CUSMST
      +000002*                          CUSTOMER MASTER RECORD                            CUSMST
      +000003*THE KEY DEFINITIONS FOR RECORD FORMAT  CUSMST                               CUSMST
      +000004*  NUMBER              NAME               RETRIEVAL     TYPE    ALTSEQ      CUSMST
      +000005*   0001   CUST                            ASCENDING      AN      NO        CUSMST
   53 +000006       05  CUSMST.                                                          CUSMST
   54 +000007           06 CUST              PIC X(5).                                     CUSMST
      +000008*                              CUSTOMER NUMBER                               CUSMST
   55 +000009           06 NAME              PIC X(25).                                    CUSMST
      +000010*                              CUSTOMER NAME                                 CUSMST
   56 +000011           06 ADDR              PIC X(20).                                    CUSMST
      +000012*                              CUSTOMER ADDRESS                              CUSMST
   57 +000013           06 CITY              PIC X(20).                                    CUSMST
      +000014*                              CUSTOMER CITY                                 CUSMST
   58 +000015           06 STATE             PIC X(2).                                     CUSMST
      +000016*                              STATE                                         CUSMST
   59 +000017           06 ZIP               PIC S9(5)        COMP-3.                      CUSMST
      +000018*                              ZIP CODE                                      CUSMST
   60 +000019           06 SRHCOD            PIC X(6).                                     CUSMST
      +000020*                              CUSTOMER NUMBER SEARCH CODE                   CUSMST
   61 +000021           06 CUSTYP            PIC S9(1)        COMP-3.                      CUSMST
      +000022*                              CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT    CUSMST
   62 +000023           06 ARBAL             PIC S9(6)V9(2)   COMP-3.                      CUSMST
      +000024*                              ACCOUNTS REC. BALANCE                         CUSMST
   63 +000025           06 ORDBAL            PIC S9(6)V9(2)   COMP-3.                      CUSMST
      +000026*                              A/R AMT. IN ORDER FILE                        CUSMST
   64 +000027           06 LSTAMT            PIC S9(6)V9(2)   COMP-3.                      CUSMST
      +000028*                              LAST AMT. PAID IN A/R                         CUSMST
   65 +000029           06 LSTDAT            PIC S9(6)        COMP-3.                      CUSMST
      +000030*                              LAST DATE PAID IN A/R                         CUSMST
   66 +000031           06 CRDLMT            PIC S9(6)V9(2)   COMP-3.                      CUSMST
      +000032*                              CUSTOMER CREDIT LIMIT                         CUSMST
   67 +000033           06 SLSYR             PIC S9(8)V9(2)   COMP-3.                      CUSMST
      +000034*                              CUSTOMER SALES THIS YEAR                      CUSMST
   68 +000035           06 SLSLYR            PIC S9(8)V9(2)   COMP-3.                      CUSMST
      +000036*                              CUSTOMER SALES LAST YEAR                      CUSMST
   69  003500
   70  003600 WORKING-STORAGE SECTION.
```

The Data Division for this example program is shown after compilation to illustrate the data structures generated by the COPY statements, DDS formats.

Figure 35 (Part 5 of 6). Example of a TRANSACTION Inquiry Program Using a Single Display Device

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   71  003700 01  ONE                                PIC 1 VALUE B"1".
   72  003800 01  CM-STATUS                          PIC X(2).
   73  003900 01  WS-CONTROL.
   74  004000     02  WS-IND                         PIC X(2).
   75  004100     02  WS-FORMAT                      PIC X(10).
   76  004200 PROCEDURE DIVISION.
       004300 BEGIN.
   77  004400     OPEN I-O CUST-DISPLAY, INPUT CUST-MASTER.
   78  004500     MOVE ZERO TO IN99 OF CUSPMT-O.
       004600 LOOP.
   79  004700     WRITE DISP-REC FORMAT IS "CUSPMT".
   80  004800     READ CUST-DISPLAY RECORD.
   81  004900     IF IN15 OF CUSPMT-I
       005000       IS EQUAL TO ONE
   82  005100        THEN GO TO FINIS.
   83  005200     MOVE CUST OF CUSPMT-I TO CUST OF CUSMST.
   84  005300     READ CUST-MASTER RECORD.
   85  005400     IF CM-STATUS IS NOT EQUAL "00" THEN
   86  005500         MOVE ONE TO IN99 OF CUSPMT-O,  GO TO LOOP.
   88  005600     MOVE CORRESPONDING CUSMST TO CUSFLDS-O.
   89  005700     WRITE DISP-REC FORMAT IS "CUSFLDS".
   90  005800     READ CUST-DISPLAY RECORD.
   91  005900     IF IN15 OF CUSFLDS-I
       006000     IS EQUAL TO ONE
   92  006100     THEN GO TO FINIS.
   93  006200     MOVE ZERO TO IN99 OF CUSPMT-O.
   94  006300     GO TO LOOP.
       006400 FINIS.
   95  006500     CLOSE CUST-DISPLAY, CUST-MASTER.
       006600 RETURN-TO-CALLER.
   96  006700     EXIT PROGRAM.
                        * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 35 (Part 6 of 6). Example of a TRANSACTION Inquiry Program Using a Single Display Device

The WRITE operation in statement 79 writes the CUSPMT record to the display. This record prompts the user to enter a customer number. If the user enters a customer number and presses the Enter key, the next READ operation then reads the record back into the program.

The READ operation in statement 84 uses the customer number (CUST) field to retrieve the corresponding CUSMST record from the CUSMSTP file. If no record is found in the CUSMSTP file, indicator 99 is set on. The GO TO operation in statement 86, which is processed when indicator 99 is set on, causes the program to branch back to the beginning. The message 'Customer number not found' is displayed because it is conditioned by indicator 99 in the DDS for the file, and the keyboard is locked. The user must press the Reset key in response to this message to unlock the keyboard. The user can then enter another customer number.

If the READ operation retrieves a record from the CUSMSTP file, the WRITE operation writes the CUSFLDS record to the display work station. This record contains the customer's name, address, and accounts receivable balance.

The user then presses the Enter key, and the program branches back to the beginning of the calculations. The user can enter another customer number or end the program. To end the program, the user presses the function key 01, which sets on indicator 15 in the program.

When indicator 15 is on, the program closes all files and processes the EXIT PROGRAM statement, which causes the program to return control to the calling COBOL program.

This is the initial display written by the WRITE operation in statement 82:

```
Customer Master Inquiry

Customer Number  _____

Use F1 to end program, use enter key to return to prompt screen
```

This display appears if a record is found in the CUSMSTP file for the customer number entered in response to the first display:

```
Customer Master Inquiry

Customer Number  10000

Use F1 to end program, use enter key to return to prompt screen

Name    EXAMPLE WHOLESALERS
Address 3561 60TH STREET
City    MOLINE
State   IL        Zipcode  61265
A/R balance       137.02
```

This display appears if the CUSMSTP file does not contain a record for the customer number entered in response to the first display:

```
Customer Master Inquiry

Customer Number  10000

Use F1 to end program, use enter key to return to prompt screen



Customer number not found  press reset, then enter valid number
```

Figure 36 on page 153 shows an example order inquiry program, ORD220, that uses subfiles. The associated DDS is also shown, except for the DDS for the customer master file, CUSMSTP. Refer to Figure 35 on page 146 for the DDS for CUSMSTP.

ORD220 displays all the detail order records for the requested order number. The program prompts the user to enter the order number that is to be reviewed. The order number is checked against the order header file, ORDHDRP. If the order number exists, the customer number accessed from the order header file is checked against the customer master file, CUSMSTP. All order detail records in ORDDTLP for the requested order are read and written to the subfile. A write for the subfile control record format is processed, and the detail order records in the subfile are displayed on the screen for the user to review. The program is ended by pressing function key 12.

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7764-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Not (N)	Conditioning: Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (b/R/K/S/I/O)	Reserved	Name	Reference (R)	Length	Data Type (b/A/P/S/B/A/S/I/X/Y/N/I/W)	Decimal Positions	Usage (b/O/I/B/N/M)	Location: Line	Location: Pos	Functions
	A	*	*	PHYSICAL					R		ORDHDRP		ORDER		HEADER				FILE
	A										ORDHDR								TEXT('ORDER HEADER RECORD')
	A										CUST		5						TEXT('CUSTOMER NUMBER')
	A										ORDER		5		00				TEXT('ORDER NUMBER')
	A										ORDDAT		6		00				TEXT('DATE ORDER WAS ENTERED')
	A										CUSORD		15						TEXT('CUSTOMER PURCHASE ORDER +
	A																		NUMBER')
	A										SHPVIA		15						TEXT('SHIPPING INSTRUCTIONS')
	A										ORDSTS		1		00				TEXT('ORDER STATUS 1PCS 2CNT 3CHK +
	A																		4RDY 5PRT 6PCK')
	A										OPRNAM		10						TEXT('OPERATOR NAME WHO ENTERED +
	A																		THE ORDER')
	A										ORDAMT		8		02				TEXT('TOTAL DOLLAR AMOUNT OF +
	A																		THE ORDER')
	A										CUSTYP		1		00				TEXT('CUSTOMER TYPE 1=GOV 2=SCH +
	A																		3=BUS 4=PVT 5=OT')
	A										INVNUM		5		00				TEXT('INVOICE NUMBER')
	A										PRTDAT		6		00				TEXT('DATE ORDER WAS PRINTED')
	A										OPNSTS		1		00				TEXT('ORDER OPEN STATUS 1=OPEN +
	A																		2=CLOSE 3-CANCEL')
	A										TOTLIN		3		00				TEXT('TOTAL LINE ITEMS IN ORDER')
	A										ACTMTH		2		00				TEXT('ACCOUNTING MONTH OF SALE')
	A										ACTYR		2		00				TEXT('ACCOUNTING YEAR OF SALE')
	A										STATE		2						TEXT('STATE')
	A										AMPAID		8		02				TEXT('TOTAL DOLLAR AMOUNT PAID')
	A								K		ORDER								
	A																		
	A																		
	A																		

Figure 36 (Part 1 of 12). Example Order Inquiry Program

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

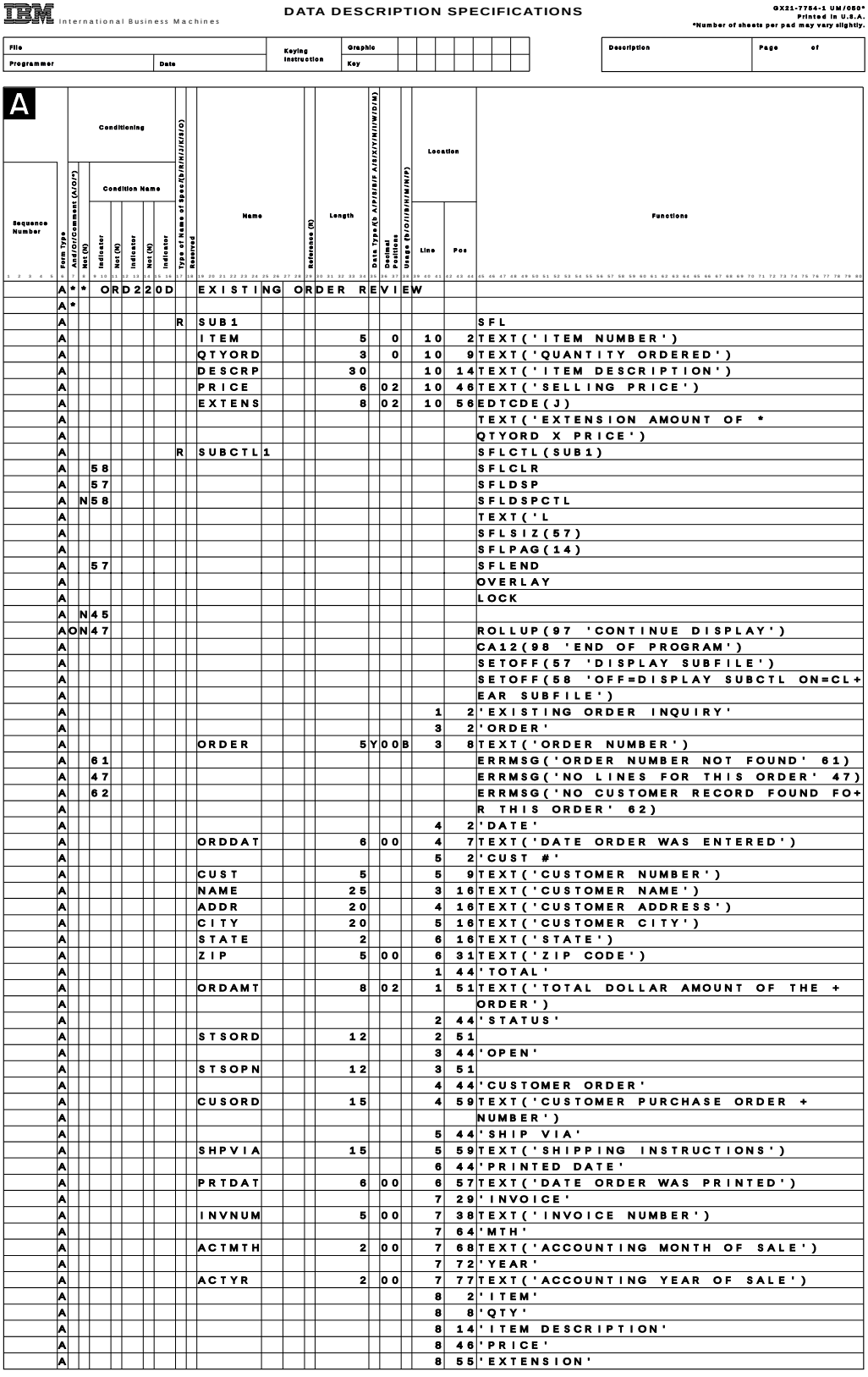
GX21-7754-1 UM/050*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Not (N)	Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Type of Name or Spec (b/R/H/J/N/S/O)	Reserved	Name	Reference (N)	Length	Data Type (b/A/P/S/B/F/A/S/X/Y/N/J/N/O/M)	Decimal Positions	Usage (b/O/I/B/H/M/N/P)	Location: Line	Pos	Functions
1 2 3 4 5	6	7	8	9 10	11	12 13	14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
	A	*	*	P H Y S I C A L							ORDDTLP		ORDER		DET	AIL			FILE
	A																		TEXT('WAREHOUSE LOCATION')
	A	*																	
	A								R		ORDDTL								TEXT('ORDER DETAIL RECORD')
	A	*																	
	A										CUST		5						CHECK(MF)
	A																		COLHDG('CUSTOMER' 'NUMBER')
	A	*																	
	A										ORDER		5		0				COLHDG('ORDER' 'NUMBER')
	A	*																	
	A										LINNUM		3		0				
	A																		COLHDG('LINE' 'NO.')
	A																		TEXT('LINE NUMBER OF LINE IN ORDER'+
	A																		)
	A	*																	
	A										ITEM		5		0				CHECK(M10)
	A																		COLHDG('ITEM' 'NUMBER')
	A										QTYORD		3		0				
	A																		COLHDG('QUANTITY' 'ORDERED')
	A																		TEXT('QUANTITY ORDERED')
	A	*																	
	A										DESCRP		30						COLHDG('ITEMDESCRIPTION')
	A	*																	
	A										PRICE		6		2				CMP(GT 0)
	A																		COLHDG('PRICE')
	A																		TEXT('SELLING PRICE')
	A																		EDTCDE(J)
	A										EXTENS		8		2				COLHDG('EXTENSION')
	A																		TEXT('EXTENSION AMOUNT OF QTYORD X +
	A																		PRICE')
	A	*																	
	A										WHSLOC		3						CHECK(MF)
	A																		COLHDG('BIN' 'NO.')
	A	*																	
	A										ORDDAT		6		0				TEXT('DATE ORDER WAS +
	A																		ENTERED')
	A										CUSTYP		1		0				RANGE(1 5)
	A																		COLHDG('CUST' 'TYPE')
	A																		TEXT('CUSTOMER TYPE 1=GOV 2=SCH +
	A																		3=BUS 4=PVT 5=OT')
	A	*																	
	A										STATE		2						CHECK(MF)
	A																		COLHDG('STATE')
	A	*																	
	A	*																	
	A										ACTMTH		2		0				COLHDG('ACCT' 'MTH')
	A																		TEXT('ACCOUNTING MONTH OF SALE')
	A	*																	
	A										ACTYR		2		0				COLHDG('ACCT' 'YEAR')
	A																		TEXT('ACCOUNTING YEAR OF SALE')
	A								K		ORDER								
	A								K		LINNUM								
	A																		

Figure 36 (Part 2 of 12). Example Order Inquiry Program



IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7754-1 UM/050*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page	of
Programmer	Date		Key										

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Not (N)	Condition Name: Indicator	Condition Name: Not (N)	Condition Name: Indicator	Condition Name: Not (N)	Condition Name: Indicator	Type of Name or Spec (B/E/H/J/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (b A/P/S/B/F A/S/X/T/Y/H/I/W/D/M)	Decimal Positions	Usage (b/O/I/B/H/M/N/P)	Location: Line	Location: Pos	Functions
	A	*	*	ORD220D							EXISTING ORDER REVIEW								
	A	*																	
	A								R		SUB1								SFL
	A										ITEM		5		0		10	2	TEXT('ITEM NUMBER')
	A										QTYORD		3		0		10	9	TEXT('QUANTITY ORDERED')
	A										DESCRP		30				10	14	TEXT('ITEM DESCRIPTION')
	A										PRICE		6		02		10	46	TEXT('SELLING PRICE')
	A										EXTENS		8		02		10	56	EDTCDE(J)
	A																		TEXT('EXTENSION AMOUNT OF *
	A																		QTYORD X PRICE')
	A								R		SUBCTL1								SFLCTL(SUB1)
	A			58															SFLCLR
	A			57															SFLDSP
	A		N	58															SFLDSPCTL
	A																		TEXT('L
	A																		SFLSIZ(57)
	A																		SFLPAG(14)
	A			57															SFLEND
	A																		OVERLAY
	A																		LOCK
	A		N	45															
	A	O	N	47															ROLLUP(97 'CONTINUE DISPLAY')
	A																		CA12(98 'END OF PROGRAM')
	A																		SETOFF(57 'DISPLAY SUBFILE')
	A																		SETOFF(58 'OFF=DISPLAY SUBCTL ON=CL+
	A																		EAR SUBFILE')
	A																1	2	'EXISTING ORDER INQUIRY'
	A																3	2	'ORDER'
	A										ORDER		5	Y	00	B	3	8	TEXT('ORDER NUMBER')
	A			61															ERRMSG('ORDER NUMBER NOT FOUND' 61)
	A			47															ERRMSG('NO LINES FOR THIS ORDER' 47)
	A			62															ERRMSG('NO CUSTOMER RECORD FOUND FO+
	A																		R THIS ORDER' 62)
	A																4	2	'DATE'
	A										ORDDAT		6		00		4	7	TEXT('DATE ORDER WAS ENTERED')
	A																5	2	'CUST #'
	A										CUST		5				5	9	TEXT('CUSTOMER NUMBER')
	A										NAME		25				3	16	TEXT('CUSTOMER NAME')
	A										ADDR		20				4	16	TEXT('CUSTOMER ADDRESS')
	A										CITY		20				5	16	TEXT('CUSTOMER CITY')
	A										STATE		2				6	16	TEXT('STATE')
	A										ZIP		5		00		6	31	TEXT('ZIP CODE')
	A																1	44	'TOTAL'
	A										ORDAMT		8		02		1	51	TEXT('TOTAL DOLLAR AMOUNT OF THE +
	A																		ORDER')
	A																2	44	'STATUS'
	A										STSORD		12				2	51	
	A																3	44	'OPEN'
	A										STSOPN		12				3	51	
	A																4	44	'CUSTOMER ORDER'
	A										CUSORD		15				4	59	TEXT('CUSTOMER PURCHASE ORDER +
	A																		NUMBER')
	A																5	44	'SHIP VIA'
	A										SHPVIA		15				5	59	TEXT('SHIPPING INSTRUCTIONS')
	A																6	44	'PRINTED DATE'
	A										PRTDAT		6		00		6	57	TEXT('DATE ORDER WAS PRINTED')
	A																7	29	'INVOICE'
	A										INVNUM		5		00		7	38	TEXT('INVOICE NUMBER')
	A																7	64	'MTH'
	A										ACTMTH		2		00		7	68	TEXT('ACCOUNTING MONTH OF SALE')
	A																7	72	'YEAR'
	A										ACTYR		2		00		7	77	TEXT('ACCOUNTING YEAR OF SALE')
	A																8	2	'ITEM'
	A																8	8	'QTY'
	A																8	14	'ITEM DESCRIPTION'
	A																8	46	'PRICE'
	A																8	55	'EXTENSION'

Figure 36 (Part 3 of 12). Example Order Inquiry Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.     EXMPLE773.
       000300*    EXAMPLE ORDER INQUIRY PROGRAM
    3  000400 AUTHOR.       PROGRAMMER NAME.
    4  000500 INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
    5  000600 DATE-WRITTEN. 08/30/88.
    6  000700 DATE-COMPILED.  08/30/88 13:54:13   .
    7  000800 ENVIRONMENT DIVISION.
    8  000900 CONFIGURATION SECTION.
    9  001000 SOURCE-COMPUTER. IBM-S38.
   10  001100 OBJECT-COMPUTER. IBM-S38.
   11  001200 INPUT-OUTPUT SECTION.
   12  001300 FILE-CONTROL.
   13  001400     SELECT ORDER-HEADER-FILE
   14  001500         ASSIGN TO DATABASE-ORDHEADE
   15  001600         ORGANIZATION IS INDEXED
   16  001700         ACCESS MODE IS RANDOM
   17  001800         RECORD KEY IS ORDERN OF ORDER-HEADER-RECORD.
   18  001900     SELECT ORDER-DETAIL-FILE
   19  002000         ASSIGN TO DATABASE-ORDDETAIL
   20  002100         ORGANIZATION IS INDEXED
   21  002200         ACCESS IS DYNAMIC
   22  002300         RECORD KEY IS ORDER-DETAIL-RECORD-KEY.
   23  002400     SELECT CUSTOMER-MASTER-FILE
   24  002500         ASSIGN TO DATABASE-CUSMSTP
   25  002600         ORGANIZATION IS INDEXED
   26  002700         ACCESS IS RANDOM
   27  002800         RECORD KEY IS CUST OF CUSTOMER-MASTER-RECORD.
   28  002900     SELECT EXISTING-ORDER-DISPLAY-FILE
   29  003000         ASSIGN TO WORKSTATION-ORD220D
   30  003100         ORGANIZATION IS TRANSACTION
   31  003200         ACCESS IS DYNAMIC
   32  003300         RELATIVE KEY IS SUBFILE-RECORD-NUMBER
   33  003400         FILE STATUS IS STATUS-CODE-ONE.
   34  003500 DATA DIVISION.
   35  003600 FILE SECTION.
   36  003700 FD  ORDER-HEADER-FILE
   37  003800     LABEL RECORDS ARE STANDARD.
   38  003900 01  ORDER-HEADER-RECORD.
   39  004000     COPY DDS-ORDHDR OF ORDHEADE.
      +000001*    I-O FORMAT:ORDHDR     FROM FILE ORDHEADE   OF LIBRARY EXMPLIB           ORDHDR
      +000002*                          ORDER HEADER RECORD                               ORDHDR
      +000003*THE KEY DEFINITIONS FOR RECORD FORMAT  ORDHDR                               ORDHDR
      +000004*  NUMBER               NAME               RETRIEVAL     TYPE    ALTSEQ     ORDHDR
      +000005*   0001   ORDERN                           ASCENDING   SIGNED     NO        ORDHDR
   40 +000006       05  ORDHDR.                                                          ORDHDR
   41 +000007           06 CUST             PIC X(5).                                    ORDHDR
      +000008*                             CUSTOMER NUMBER                               ORDHDR
   42 +000009           06 ORDERN            PIC S9(5)        COMP-3.                    ORDHDR
      +000010*                             ORDER NUMBER                                  ORDHDR
   43 +000011           06 ORDDAT            PIC S9(6)        COMP-3.                    ORDHDR
      +000012*                             DATE ORDER ENTERED                            ORDHDR
   44 +000013           06 CUSORD            PIC X(15).                                   ORDHDR
      +000014*                             CUSTOMER PURCHASE ORDER NUMBER                ORDHDR
   45 +000015           06 SHPVIA            PIC X(15).                                   ORDHDR
```

Figure 36 (Part 4 of 12). Example Order Inquiry Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      +000016*                       SHIPPING INSTRUCTIONS                                 ORDHDR
   46 +000017           06 ORDSTS           PIC S9(1)        COMP-3.                      ORDHDR
      +000018*                       ORDER STATUS 1PCS 2CNT  3CHK 4RDY 5PRT 6PC           ORDHDR
   47 +000019           06 OPRNAM           PIC X(10).                                    ORDHDR
      +000020*                       OPERATOR WHO ENTERED ORD                              ORDHDR
   48 +000021           06 ORDAMT           PIC S9(6)V9(2)   COMP-3.                      ORDHDR
      +000022*                       DOLLAR AMOUNT OF ORDER                                ORDHDR
   49 +000023           06 CUSTYP           PIC S9(1)        COMP-3.                      ORDHDR
      +000024*                       CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT            ORDHDR
   50 +000025           06 INVNUM           PIC S9(5)        COMP-3.                      ORDHDR
      +000026*                       INVOICE NUMBER                                        ORDHDR
   51 +000027           06 PRTDAT           PIC S9(6)        COMP-3.                      ORDHDR
      +000028*                       DATE ORDER WAS PRINTED                                ORDHDR
   52 +000029           06 OPNSTS           PIC S9(1)        COMP-3.                      ORDHDR
      +000030*                       ORDER OPEN STATUS 1=OPEN 2= CLOSE 3=CANCEL            ORDHDR
   53 +000031           06 TOTLIN           PIC S9(3)        COMP-3.                      ORDHDR
      +000032*                       TOTAL LINE ITEMS IN ORDER                             ORDHDR
   54 +000033           06 ACTMTH           PIC S9(2)        COMP-3.                      ORDHDR
      +000034*                       ACCOUNTING MONTH OF SALE                              ORDHDR
   55 +000035           06 STATE            PIC X(2).                                     ORDHDR
      +000036*                       STATE                                                 ORDHDR
   56 +000037           06 AMPAID           PIC S9(6)V9(2)   COMP-3.                      ORDHDR
      +000038*                       AMOUNT PAID                                           ORDHDR
   57  004100
   58  004200 FD  ORDER-DETAIL-FILE
   59  004300     LABEL RECORDS ARE STANDARD.
   60  004400 01  ORDER-DETAIL-RECORD.
   61  004500     COPY DDS-ORDDTL OF ORDDETAIL.
      +000001*    I-O FORMAT:ORDDTL     FROM FILE ORDDETAIL  OF LIBRARY EXMPLIB           ORDDTL
      +000002*                       ORDER DETAIL RECORD                                   ORDDTL
      +000003*THE KEY DEFINITIONS FOR RECORD FORMAT  ORDDTL                                 ORDDTL
      +000004*  NUMBER              NAME              RETRIEVAL     TYPE    ALTSEQ       ORDDTL
      +000005*   0001   ORDERN                         ASCENDING   SIGNED     NO         ORDDTL
      +000006*   0002   LINNUM                         ASCENDING   SIGNED     NO         ORDDTL
   62 +000007       05  ORDDTL.                                                          ORDDTL
   63 +000008           06 CUST             PIC X(5).                                     ORDDTL
      +000009*                       CUSTOMER NUMBER                                       ORDDTL
   64 +000010           06 ORDERN           PIC S9(5)        COMP-3.                      ORDDTL
      +000011*                       ORDER NUMBER                                          ORDDTL
   65 +000012           06 LINNUM           PIC S9(3)        COMP-3.                      ORDDTL
      +000013*                       LINE NUMBER OF LINE IN ORDER                          ORDDTL
   66 +000014           06 ITEM             PIC S9(5)        COMP-3.                      ORDDTL
      +000015*                       ITEM NUMBER                                           ORDDTL
   67 +000016           06 QTYORD           PIC S9(3)        COMP-3.                      ORDDTL
      +000017*                       QUANTITY ORDERED                                      ORDDTL
   68 +000018           06 DESCRP           PIC X(30).                                    ORDDTL
      +000019*                       ITEM DESCRIPTION                                      ORDDTL
   69 +000020           06 PRICE            PIC S9(4)V9(2)   COMP-3.                      ORDDTL
      +000021*                       SELLING PRICE                                         ORDDTL
   70 +000022           06 EXTENS           PIC S9(6)V9(2)   COMP-3.                      ORDDTL
      +000023*                       EXTENSION AMOUNT OF QTYORD X PRICE                    ORDDTL
   71 +000024           06 WHSLOC           PIC X(3).                                     ORDDTL
      +000025*                       BIN NO.                                               ORDDTL
   72 +000026           06 ORDDAT           PIC S9(6)        COMP-3.                      ORDDTL
      +000027*                       DATE ORDER WAS ENTERED                                ORDDTL
```

Figure 36 (Part 5 of 12). Example Order Inquiry Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   73 +000028           06 CUSTYP            PIC S9(1)        COMP-3.                      ORDDTL
      +000029*                             CUSTOMER TYPE 1=GOV 2=SCH   3=BUS 4=PVT 5=       ORDDTL
   74 +000030           06 STATE             PIC X(2).                                     ORDDTL
      +000031*                             STATE                                         ORDDTL
   75 +000032           06 ACTMTH            PIC S9(2)        COMP-3.                      ORDDTL
      +000033*                             ACCOUNTING MONTH OF SALE                      ORDDTL
   76 +000034           06 ACTYR             PIC S9(2)        COMP-3.                      ORDDTL
      +000035*                             ACCOUNTING YEAR OF SALE                       ORDDTL
   77  004600 66  ORDER-DETAIL-RECORD-KEY RENAMES ORDERN THRU LINNUM.
   78  004700
   79  004800 FD  CUSTOMER-MASTER-FILE
   80  004900     LABEL RECORDS ARE STANDARD.
   81  005000 01  CUSTOMER-MASTER-RECORD.
   82  005100     COPY DDS-CUSMST OF CUSMSTP.
      +000001*    I-O FORMAT:CUSMST     FROM FILE CUSMSTP    OF LIBRARY EXMPLIB           CUSMST
      +000002*                          ORDER HEADER RECORD                               CUSMST
      +000003*THE KEY DEFINITIONS FOR RECORD FORMAT  CUSMST                               CUSMST
      +000004*  NUMBER              NAME               RETRIEVAL     TYPE    ALTSEQ       CUSMST
      +000005*   0001   CUST                            ASCENDING      AN      NO         CUSMST
   83 +000006       05  CUSMST.                                                          CUSMST
   84 +000007           06 CUST              PIC X(5).                                     CUSMST
      +000008*                             CUSTOMER NUMBER                               CUSMST
   85 +000009           06 NAME              PIC X(25).                                    CUSMST
      +000010*                             CUSTOMER NAME                                 CUSMST
   86 +000011           06 ADDR              PIC X(20).                                    CUSMST
      +000012*                             CUSTOMER ADDRESS                              CUSMST
   87 +000013           06 CITY              PIC X(20).                                    CUSMST
      +000014*                             CUSTOMER CITY                                 CUSMST
   88 +000015           06 STATE             PIC X(2).                                     CUSMST
      +000016*                             STATE                                         CUSMST
   89 +000017           06 ZIP               PIC S9(5)        COMP-3.                      CUSMST
      +000018*                             ZIP CODE                                      CUSMST
   90 +000019           06 SRHCOD            PIC X(6).                                     CUSMST
      +000020*                             CUSTOMER NUMBER SEARCH CODE                   CUSMST
   91 +000021           06 CUSTYP            PIC S9(1)        COMP-3.                      CUSMST
      +000022*                             CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT    CUSMST
   92 +000023           06 ARBAL             PIC S9(6)V9(2)   COMP-3.                      CUSMST
      +000024*                             ACCOUNTS REC. BALANCE                         CUSMST
   93 +000025           06 ORDBAL            PIC S9(6)V9(2)   COMP-3.                      CUSMST
      +000026*                             A/R AMT. IN ORDER FILE                        CUSMST
   94 +000027           06 LSTAMT            PIC S9(6)V9(2)   COMP-3.                      CUSMST
      +000028*                             LAST AMT. PAID IN A/R                         CUSMST
   95 +000029           06 LSTDAT            PIC S9(6)        COMP-3.                      CUSMST
      +000030*                             LAST DATE PAID IN A/R                         CUSMST
   96 +000031           06 CRDLMT            PIC S9(6)V9(2)   COMP-3.                      CUSMST
      +000032*                             CUSTOMER CREDIT LIMIT                         CUSMST
   97 +000033           06 SLSYR             PIC S9(8)V9(2)   COMP-3.                      CUSMST
      +000034*                             CUSTOMER SALES THIS YEAR                      CUSMST
   98 +000035           06 SLSLYR            PIC S9(8)V9(2)   COMP-3.                      CUSMST
      +000036*                             CUSTOMER SALES LAST YEAR                      CUSMST
   99  005200
  100  005300 FD  EXISTING-ORDER-DISPLAY-FILE
  101  005400     LABEL RECORDS ARE OMITTED.
  102  005500 01  EXISTING-ORDER-DISPLAY-RECORD.
  103  005600     COPY DDS-ALL-FORMATS OF ORD220D.
```

Figure 36 (Part 6 of 12). Example Order Inquiry Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  104 +000001       05  ORD220D-RECORD PIC X(171).                                        <-ALL-FMTS
      +000002*    I-O FORMAT:SUB1       FROM FILE ORD220D    OF LIBRARY EXMPLIB          <-ALL-FMTS
      +000003*                                                                           <-ALL-FMTS
  105 +000004       05  SUB1          REDEFINES ORD220D-RECORD.                         <-ALL-FMTS
  106 +000005           06 ITEM             PIC S9(5).                                  <-ALL-FMTS
      +000006*                            ITEM NUMBER                                   <-ALL-FMTS
  107 +000007           06 QTYORD           PIC S9(3).                                  <-ALL-FMTS
      +000008*                            QUANTITY ORDERED                              <-ALL-FMTS
  108 +000009           06 DESCRP           PIC X(30).                                  <-ALL-FMTS
      +000010*                            ITEM DESCRIPTION                              <-ALL-FMTS
  109 +000011           06 PRICE            PIC S9(4)V9(2).                             <-ALL-FMTS
      +000012*                            SELLING PRICE                                 <-ALL-FMTS
  110 +000013           06 EXTENS           PIC S9(6)V9(2).                             <-ALL-FMTS
      +000014*                            EXTENSION AMOUNT OF QTYORD X PRICE            <-ALL-FMTS
      +000015*  INPUT FORMAT:SUBCTL1    FROM FILE ORD220D    OF LIBRARY EXMPLIB          <-ALL-FMTS
      +000016*                                                                           <-ALL-FMTS
  111 +000017       05  SUBCTL1-I     REDEFINES ORD220D-RECORD.                         <-ALL-FMTS
  112 +000018           06 SUBCTL1-I-INDIC.                                             <-ALL-FMTS
  113 +000019                07 IN97        PIC 1  INDIC 97.                           <-ALL-FMTS
      +000020*                            CONTINUE DISPLAY                              <-ALL-FMTS
  114 +000021                07 IN98        PIC 1  INDIC 98.                           <-ALL-FMTS
      +000022*                            END OF PROGRAM                                <-ALL-FMTS
  115 +000023                07 IN57        PIC 1  INDIC 57.                           <-ALL-FMTS
      +000024*                            DISPLAY SUBFILE                               <-ALL-FMTS
  116 +000025                07 IN58        PIC 1  INDIC 58.                           <-ALL-FMTS
      +000026*                            OFF = DISPLAY SUBCTL1 ON = CLEAR SUBFILE      <-ALL-FMTS
  117 +000027                07 IN61        PIC 1  INDIC 61.                           <-ALL-FMTS
      +000028*                            ORDER NUMBER NOT FOUND                        <-ALL-FMTS
  118 +000029                07 IN47        PIC 1  INDIC 47.                           <-ALL-FMTS
      +000030*                            NO LINE FOR THIS ORDER                        <-ALL-FMTS
  119 +000031                07 IN62        PIC 1  INDIC 62.                           <-ALL-FMTS
      +000032*                            NO CUSTOMER RECORD                            <-ALL-FMTS
  120 +000033           06 ORDERN           PIC S9(5).                                  <-ALL-FMTS
      +000034*                            ORDER NUMBER                                  <-ALL-FMTS
      +000035* OUTPUT FORMAT:SUBCTL1    FROM FILE ORD220D    OF LIBRARY EXMPLIB          <-ALL-FMTS
      +000036*                                                                           <-ALL-FMTS
  121 +000037       05  SUBCTL1-O     REDEFINES ORD220D-RECORD.                         <-ALL-FMTS
  122 +000038           06 SUBCTL1-O-INDIC.                                             <-ALL-FMTS
  123 +000039                07 IN58        PIC 1  INDIC 58.                           <-ALL-FMTS
      +000040*                            OFF = DISPLAY SUBCTL1 ON = CLEAR SUBFILE      <-ALL-FMTS
  124 +000041                07 IN57        PIC 1  INDIC 57.                           <-ALL-FMTS
      +000042*                            DISPLAY SUBFILE                               <-ALL-FMTS
  125 +000043                07 IN45        PIC 1  INDIC 45.                           <-ALL-FMTS
  126 +000044                07 IN47        PIC 1  INDIC 47.                           <-ALL-FMTS
      +000045*                            NO LINE FOR THIS ORDER                        <-ALL-FMTS
  127 +000046                07 IN61        PIC 1  INDIC 61.                           <-ALL-FMTS
      +000047*                            ORDER NUMBER NOT FOUND                        <-ALL-FMTS
  128 +000048                07 IN62        PIC 1  INDIC 62.                           <-ALL-FMTS
      +000049*                            NO CUSTOMER RECORD                            <-ALL-FMTS
  129 +000050           06 ORDERN           PIC S9(5).                                  <-ALL-FMTS
      +000051*                            ORDER NUMBER                                  <-ALL-FMTS
  130 +000052           06 ORDDAT           PIC S9(6).                                  <-ALL-FMTS
      +000053*                            DATE ORDER WAS ENTERED                        <-ALL-FMTS
  131 +000054           06 CUST             PIC X(5).                                   <-ALL-FMTS
      +000055*                            CUSTOMER NUMBER                               <-ALL-FMTS
```

Figure 36 (Part 7 of 12). Example Order Inquiry Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  132 +000056           06 NAME              PIC X(25).                                    <-ALL-FMTS
      +000057*                              CUSTOMER NAME                                 <-ALL-FMTS
  133 +000058           06 ADDR              PIC X(20).                                    <-ALL-FMTS
      +000059*                              CUSTOMER ADDRESS                              <-ALL-FMTS
  134 +000060           06 CITY              PIC X(20).                                    <-ALL-FMTS
      +000061*                              CUSTOMER CITY                                 <-ALL-FMTS
  135 +000062           06 STATE             PIC X(2).                                     <-ALL-FMTS
      +000063*                              CUSTOMER STATE                                <-ALL-FMTS
  136 +000064           06 ZIP               PIC S9(5).                                    <-ALL-FMTS
      +000065*                              ZIP CODE                                      <-ALL-FMTS
  137 +000066           06 ORDAMT            PIC S9(6)V9(2).                               <-ALL-FMTS
      +000067*                              TOTAL AMOUNT OF ORDER                         <-ALL-FMTS
  138 +000068           06 STSORD            PIC X(12).                                    <-ALL-FMTS
  139 +000069           06 STSOPN            PIC X(12).                                    <-ALL-FMTS
  140 +000070           06 CUSORD            PIC X(15).                                    <-ALL-FMTS
      +000071*                              CUSTOMER PURCHASE ORDER NUMBER                <-ALL-FMTS
  141 +000072           06 SHPVIA            PIC X(15).                                    <-ALL-FMTS
      +000073*                              SHIPPING INSTRUCTIONS                         <-ALL-FMTS
  142 +000074           06 PRTDAT            PIC S9(6).                                    <-ALL-FMTS
      +000075*                              DATE ORDER WAS PRINTED                        <-ALL-FMTS
  143 +000076           06 INVNUM            PIC S9(5).                                    <-ALL-FMTS
      +000077*                              INVOICE NUMBER                                <-ALL-FMTS
  144 +000078           06 ACTMTH            PIC S9(2).                                    <-ALL-FMTS
      +000079*                              ACCOUNTING MONTH OF SALE                      <-ALL-FMTS
  145 +000080           06 ACTYR             PIC S9(2).                                    <-ALL-FMTS
      +000081*                              ACCOUNTING YEAR OF SALE                       <-ALL-FMTS
  146  005700
  147  005800 WORKING-STORAGE SECTION.
  148  005900 01  EXISTING-ORDER-DISPLAY-KEY.
  149  006000    05  SUBFILE-RECORD-NUMBER               PIC 9(2)
  150  006100                                        VALUE ZERO.
  151  006200
  152  006300 01  ORDER-STATUS-COMMENT-VALUES.
  153  006400    05  FILLER                              PIC X(12)
  154  006500                                        VALUE "1-IN PROCESS".
  155  006600    05  FILLER                              PIC X(12)
  156  006700                                        VALUE "2-CONTINUED ".
  157  006800    05  FILLER                              PIC X(12)
  158  006900                                        VALUE "3-CREDIT CHK".
  159  007000    05  FILLER                              PIC X(12)
  160  007100                                        VALUE "4-READY PRT ".
  161  007200    05  FILLER                              PIC X(12)
  162  007300                                        VALUE "5-PRINTED   ".
  163  007400    05  FILLER                              PIC X(12)
  164  007500                                        VALUE "6-PICKED    ".
  165  007600    05  FILLER                              PIC X(12)
  166  007700                                        VALUE "7-INVOICED  ".
  167  007800    05  FILLER                              PIC X(12)
  168  007900                                        VALUE "8-INVALID   ".
  169  008000    05  FILLER                              PIC X(12)
  170  008100                                        VALUE "9-CANCELED  ".
  171  008200
  172  008300 01  ORDER-STATUS-COMMENT-TABLE
  173  008400         REDEFINES ORDER-STATUS-COMMENT-VALUES.
  174  008500    05  ORDER-STATUS OCCURS 9 TIMES.
```

Figure 36 (Part 8 of 12). Example Order Inquiry Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  175  008600       10 ORDER-STATUS-COMMENT              PIC X(12).
  176  008700
  177  008800 01  OPEN-STATUS-COMMENT-VALUES.
  178  008900    05  FILLER                               PIC X(12)
  179  009000                                          VALUE "1-OPEN      ".
  180  009100    05  FILLER                               PIC X(12)
  181  009200                                          VALUE "2-CLOSED    ".
  182  009300    05  FILLER                               PIC X(12)
  183  009400                                          VALUE "3-CANCELED  ".
  184  009500
  185  009600 01  OPEN-STATUS-COMMENT-TABLE
  186  009700         REDEFINES OPEN-STATUS-COMMENT-VALUES.
  187  009800    05  OPEN-STATUS OCCURS 3 TIMES.
  188  009900       10  OPEN-STATUS-COMMENT              PIC X(12).
  189  010000
  190  010100 01  ERRHDL-PARAMETERS.
  191  010200    05  STATUS-CODE-ONE                      PIC X(2).
  192  010300       88  SUBFILE-IS-FULL                VALUE "9M".
  193  010400
  194  010500 01  ERRPGM-PARAMETERS.
  195  010600    05  DISPLAY-PARAMETER                    PIC X(8)
  196  010700                                          VALUE "ORD220D ".
  197  010800    05  DUMMY-ONE                            PIC X(6)
  198  010900                                          VALUE SPACES.
  199  011000    05  DUMMY-TWO                            PIC X(8)
  200  011100                                          VALUE SPACES.
  201  011200    05  STATUS-CODE-TWO.
  202  011300       10  PRIMARY                          PIC X(1).
  203  011400       10  SECONDARY                        PIC X(1).
  204  011500       10  FILLER                           PIC X(5)
  205  011600                                          VALUE SPACES.
  206  011700
  207  011800 01  SWITCH-AREA.
  208  011900    05  SW01                                 PIC 1.
  209  012000       88  NO-MORE-DETAIL-LINE-ITEMS      VALUE B"1".
  210  012100       88  MORE-DETAIL-LINE-ITEMS-EXIST   VALUE B"0".
  211  012200    05  SW02                                 PIC 1.
  212  012300       88  WRITE-DISPLAY                  VALUE B"1".
  213  012400       88  READ-DISPLAY                   VALUE B"0".
  214  012500    05  SW03                                 PIC 1.
  215  012600       88  SUBCTL1-FORMAT                 VALUE B"1".
  216  012700       88  NOT-SUBCTL1-FORMAT             VALUE B"0".
  217  012800    05  SW04                                 PIC 1.
  218  012900       88  SUB1-FORMAT                    VALUE B"1".
  219  013000       88  NOT-SUB1-FORMAT                VALUE B"0".
  220  013100
  221  013200 01  INDICATOR-AREA.
  222  013300    05  IN98                                 PIC 1 INDIC 98.
  223  013400       88  END-OF-EXISTING-ORDER-INQUIRY  VALUE B"1".
  224  013500    05  IN97                                 PIC 1 INDIC 97.
  225  013600       88  CONTINUE-DETAIL-LINES-DISPLAY  VALUE B"1".
  226  013700    05  IN62                                 PIC 1 INDIC 62.
  227  013800       88  CUSTOMER-NOT-FOUND              VALUE B"1".
  228  013900       88  CUSTOMER-EXIST                 VALUE B"0".
  229  014000    05  IN61                                 PIC 1 INDIC 61.
```

Figure 36 (Part 9 of 12). Example Order Inquiry Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  230  014100       88  ORDER-NOT-FOUND               VALUE B"1".
  231  014200       88  ORDER-EXIST                   VALUE B"0".
  232  014300    05  IN58                               PIC 1 INDIC 58.
  233  014400       88  CLEAR-SUBFILE                 VALUE B"1".
  234  014500       88  DISPLAY-SUBFILE-CONTROL       VALUE B"0".
  235  014600    05  IN57                               PIC 1 INDIC 57.
  236  014700       88  DISPLAY-SUBFILE               VALUE B"1".
  237  014800    05  IN47                               PIC 1 INDIC 47.
  238  014900       88  NO-DETAIL-LINES-FOR-ORDER     VALUE B"1".
  239  015000       88  DETAIL-LINES-FOR-ORDER-EXIST  VALUE B"0".
  240  015100    05  IN45                               PIC 1 INDIC 45.
  241  015200       88  END-OF-ORDER                  VALUE B"1".
  242  015300
  243  015400 PROCEDURE DIVISION.
       015500
       015600 DECLARATIVES.
       015700 TRANSACTION-ERROR SECTION.
       015800     USE AFTER STANDARD ERROR PROCEDURE
       015900         EXISTING-ORDER-DISPLAY-FILE.
       016000 WORK-STATION-ERROR-HANDLER.
  244  016100     IF SUBFILE-IS-FULL THEN
       016200         NEXT SENTENCE
       016300     ELSE
  245  016400         DISPLAY "WORK-STATION ERROR"   STATUS-CODE-ONE.
       016500 END DECLARATIVES.
       016600
       016700 INQUIRY-INTO-EXISTING-ORDER SECTION.
       016800 MAINLINE-ROUTINE.
  246  016900     PERFORM SET-UP-ROUTINE.
  247  017000     PERFORM EXISTING-ORDER-INQUIRY
       017100         UNTIL END-OF-EXISTING-ORDER-INQUIRY.
  248  017200     PERFORM CLEAN-UP-ROUTINE.
       017300
       017400 SET-UP-ROUTINE.
  249  017500     OPEN INPUT  ORDER-HEADER-FILE
       017600                 ORDER-DETAIL-FILE
       017700                 CUSTOMER-MASTER-FILE
       017800           I-O   EXISTING-ORDER-DISPLAY-FILE.
  250  017900     MOVE SPACES TO   CUST   OF SUBCTL1-O
       018000                      NAME   OF SUBCTL1-O
       018100                      ADDR   OF SUBCTL1-O
       018200                      CITY   OF SUBCTL1-O
       018300                      STATE  OF SUBCTL1-O
       018400                      STSORD OF SUBCTL1-O
       018500                      STSOPN OF SUBCTL1-O
       018600                      CUSORD OF SUBCTL1-O.
  251  018700     MOVE ZEROS TO    ORDERN OF SUBCTL1-O
       018800                      ORDDAT OF SUBCTL1-O
       018900                      ZIP    OF SUBCTL1-O
       019000                      ORDAMT OF SUBCTL1-O
       019100                      PRTDAT OF SUBCTL1-O
       019200                      INVNUM OF SUBCTL1-O
       019300                      ACTMTH OF SUBCTL1-O
       019400                      ACTYR  OF SUBCTL1-O.
  252  019500     MOVE B"0" TO INDICATOR-AREA.
```

Figure 36 (Part 10 of 12). Example Order Inquiry Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  253  019600     SET READ-DISPLAY
       019700         NOT-SUBCTL1-FORMAT
       019800         NOT-SUB1-FORMAT TO TRUE.
  254  019900     MOVE CORR INDICATOR-AREA TO SUBCTL1-O-INDIC.
  255  020000     WRITE EXISTING-ORDER-DISPLAY-RECORD FORMAT IS "SUBCTL1".
  256  020100     READ EXISTING-ORDER-DISPLAY-FILE RECORD.
  257  020200     MOVE CORR SUBCTL1-I-INDIC TO INDICATOR-AREA.
       020300
       020400 EXISTING-ORDER-INQUIRY.
  258  020500     IF CONTINUE-DETAIL-LINES-DISPLAY THEN
  259  020600         PERFORM READ-NEXT-ORDER-DETAIL-RECORD
  260  020700         IF MORE-DETAIL-LINE-ITEMS-EXIST THEN
  261  020800             IF ORDERN OF ORDER-DETAIL-RECORD IS NOT EQUAL TO
       020900                ORDERN OF ORDER-HEADER-RECORD THEN
  262  021000                    SET DISPLAY-SUBFILE TO TRUE
  263  021100                    SET NO-DETAIL-LINES-FOR-ORDER TO TRUE
       021200             ELSE
  264  021300                 PERFORM SUBFILE-SET-UP
       021400         ELSE
  265  021500             SET DISPLAY-SUBFILE TO TRUE
  266  021600             SET NO-DETAIL-LINES-FOR-ORDER TO TRUE
       021700     ELSE
  267  021800         PERFORM ORDER-NUMBER-VALIDATION.
  268  021900     MOVE CORR INDICATOR-AREA TO SUBCTL1-O-INDIC.
  269  022000     SET WRITE-DISPLAY TO TRUE.
  270  022100     SET SUBCTL1-FORMAT TO TRUE.
  271  022200     WRITE EXISTING-ORDER-DISPLAY-RECORD FORMAT IS "SUBCTL1".
  272  022300     READ EXISTING-ORDER-DISPLAY-FILE RECORD.
  273  022400     MOVE CORR SUBCTL1-I-INDIC TO INDICATOR-AREA.
       022500 ORDER-NUMBER-VALIDATION.
  274  022600     PERFORM READ-ORDER-HEADER-FILE.
  275  022700     IF ORDER-EXIST THEN
  276  022800         PERFORM READ-CUSTOMER-MASTER-FILE
  277  022900         IF CUSTOMER-EXIST THEN
  278  023000             PERFORM READ-FIRST-ORDER-DETAIL-RECORD
  279  023100             IF DETAIL-LINES-FOR-ORDER-EXIST THEN
  280  023200                 PERFORM SUBFILE-SET-UP
       023300             ELSE
       023400                 NEXT SENTENCE
       023500         ELSE
       023600             NEXT SENTENCE
       023700     ELSE
       023800         NEXT SENTENCE.
       023900 READ-ORDER-HEADER-FILE.
  281  024000     MOVE ORDERN OF SUBCTL1-I OF EXISTING-ORDER-DISPLAY-RECORD
       024100         TO ORDERN OF ORDER-HEADER-RECORD.
  282  024200     READ ORDER-HEADER-FILE
  283  024300         INVALID KEY SET ORDER-NOT-FOUND TO TRUE.
       024400 READ-CUSTOMER-MASTER-FILE.
  284  024500     MOVE CUST OF ORDER-HEADER-RECORD
       024600         TO CUST OF CUSTOMER-MASTER-RECORD.
  285  024700     READ CUSTOMER-MASTER-FILE
  286  024800         INVALID KEY SET CUSTOMER-NOT-FOUND TO TRUE.
       024900 READ-FIRST-ORDER-DETAIL-RECORD.
  287  025000     MOVE ORDERN OF ORDER-HEADER-RECORD
```

Figure 36 (Part 11 of 12). Example Order Inquiry Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      025100         TO ORDERN OF ORDER-DETAIL-RECORD.
  288 025200     MOVE 1 TO LINNUM OF ORDER-DETAIL-RECORD.
  289 025300     READ ORDER-DETAIL-FILE
  290 025400         INVALID KEY SET NO-DETAIL-LINES-FOR-ORDER TO TRUE.
      025500 SUBFILE-SET-UP.
  291 025600     SET CLEAR-SUBFILE TO TRUE.
  292 025700     MOVE CORR INDICATOR-AREA TO SUBCTL1-O-INDIC.
  293 025800     SET WRITE-DISPLAY TO TRUE.
  294 025900     SET SUBCTL1-FORMAT TO TRUE.
  295 026000     WRITE EXISTING-ORDER-DISPLAY-RECORD FORMAT IS "SUBCTL1".
  296 026100     SET DISPLAY-SUBFILE-CONTROL TO TRUE.
  297 026200     PERFORM BUILD-DISPLAY-SUBFILE
      026300         UNTIL NO-MORE-DETAIL-LINE-ITEMS
      026400           OR SUBFILE-IS-FULL.
  298 026500     MOVE CORR ORDHDR OF ORDER-HEADER-RECORD
      026600         TO SUBCTL1-O OF EXISTING-ORDER-DISPLAY-RECORD.
  299 026700     MOVE CORR CUSMST OF CUSTOMER-MASTER-RECORD
      026800         TO SUBCTL1-O OF EXISTING-ORDER-DISPLAY-RECORD.
  300 026900     MOVE ORDER-STATUS(ORDSTS) TO STSORD.
  301 027000     MOVE OPEN-STATUS(OPNSTS) TO STSOPN.
  302 027100     SET MORE-DETAIL-LINE-ITEMS-EXIST TO TRUE.
  303 027200     MOVE ZEROS TO SUBFILE-RECORD-NUMBER.
      027300 BUILD-DISPLAY-SUBFILE.
  304 027400     MOVE CORR ORDDTL OF ORDER-DETAIL-RECORD
      027500         TO SUB1 OF EXISTING-ORDER-DISPLAY-RECORD.
  305 027600     SET WRITE-DISPLAY TO TRUE.
  306 027700     SET SUB1-FORMAT TO TRUE.
  307 027800     ADD 1 TO SUBFILE-RECORD-NUMBER.
  308 027900     WRITE SUBFILE EXISTING-ORDER-DISPLAY-RECORD FORMAT IS "SUB1".
  309 028000     IF SUBFILE-IS-FULL THEN
  310 028100         SET DISPLAY-SUBFILE TO TRUE
      028200     ELSE
  311 028300         PERFORM READ-NEXT-ORDER-DETAIL-RECORD
  312 028400         IF NO-MORE-DETAIL-LINE-ITEMS THEN
      028500             NEXT SENTENCE
      028600         ELSE
  313 028700             IF ORDERN OF ORDER-DETAIL-RECORD IS NOT EQUAL TO
      028800                ORDERN OF ORDER-HEADER-RECORD THEN
  314 028900                 SET DISPLAY-SUBFILE TO TRUE
  315 029000                 SET NO-MORE-DETAIL-LINE-ITEMS TO TRUE
      029100             ELSE
      029200                 NEXT SENTENCE.
      029300 READ-NEXT-ORDER-DETAIL-RECORD.
  316 029400     READ ORDER-DETAIL-FILE NEXT RECORD
  317 029500         AT END SET DISPLAY-SUBFILE TO TRUE
  318 029600                SET NO-MORE-DETAIL-LINE-ITEMS TO TRUE.
      029700 CLEAN-UP-ROUTINE.
  319 029800      CLOSE     ORDER-HEADER-FILE
      029900                ORDER-DETAIL-FILE
      030000                CUSTOMER-MASTER-FILE
      030100                EXISTING-ORDER-DISPLAY-FILE.
  320 030200      STOP RUN.
                        * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 36 (Part 12 of 12). Example Order Inquiry Program

This is the initial order entry prompt display written to the workstation:

```
Existing Order Inquiry                    Total  00000000
                                          Status
Order 00000                               Open
Date 000000                               Customer order
Cust #                                    Ship via
                            00000         Printed date 000000
                          Invoice  00000                    Mth 00  Year 00
Item  Qty   Item description                Price    Extension
```

This display appears if there are detail order records for the customer whose order number was entered in the first display:

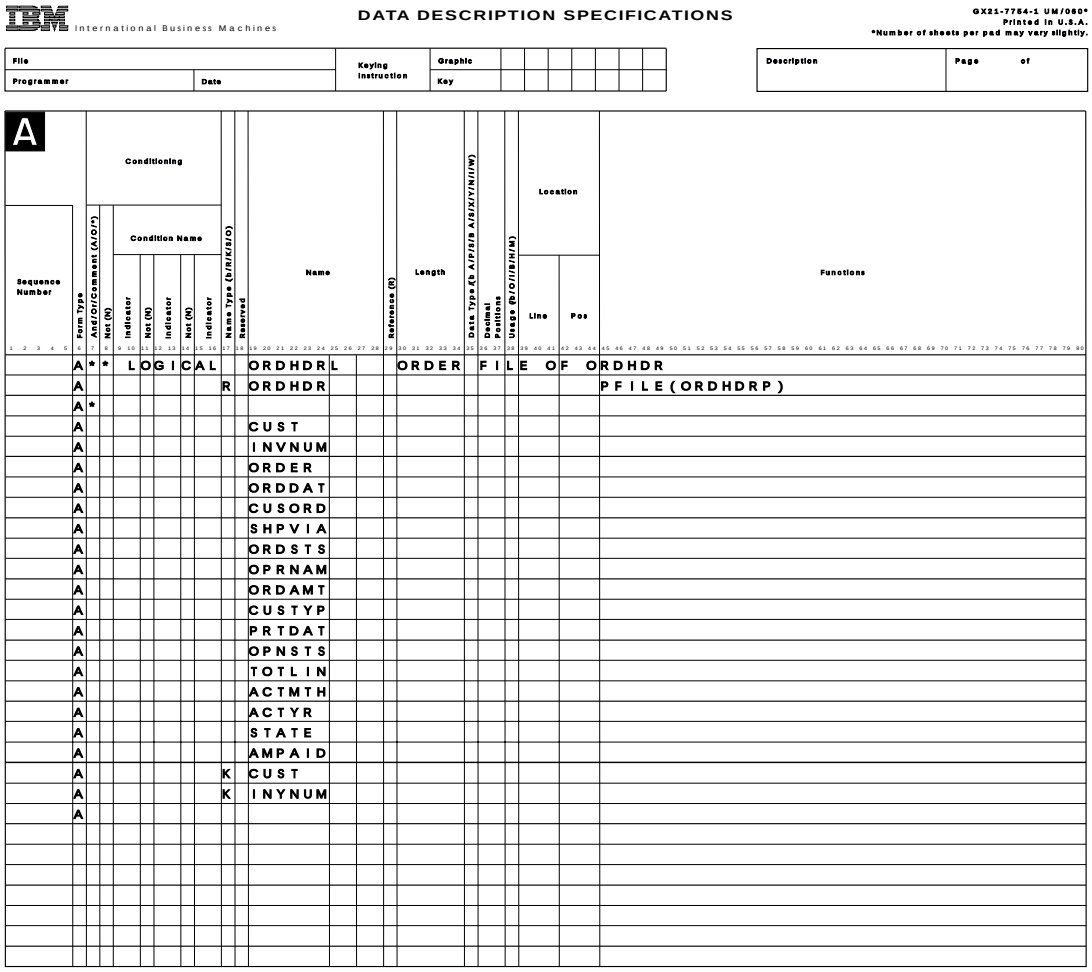
```
Existing Order Inquiry                    Total  00742656
                                          Status 7-INVOICED
Order 17924   TESTCASE HARDWARE CO        Open   2-CLOSED
Date 110587   1204 BURNSIDE DR            Customer order TESTCS17933001I
Cust # 11200  KANKAKEE                    Ship via       TRUCKCO
              IL             60901        Printed date 042578
                          Invoice  17924                    Mth 12  Year 87
Item  Qty   Item description                Price    Extension
33001  003  TORQUE WRENCH 75LB 14 INCH      009115        273.45
33100  001  TORQUE WRENCH W/GAUGE 200 LB    015777        157.77
44529  004  WOOD CHISEL - 3 1/4             006840        273.60
44958  002  POWER DRILL - 3/8 REV           008200        164.00
46102  003  WROUGHT IRON RAILING 4FTX6FT    007930        237.90
46201  001  WROUGHT IRON HAND RAIL 4X4FT    007178         71.78
47902  005  ESCUTCHEON BRASS 15X4INCHES     044488      2,224.40
48108  002  DOOR CHIME ELECTRIC 6 NOTE      104202      2,084.04
48801  004  AWNING ALUMINUM 4FT STRIPED     043002      1,720.08
48900  001  AWNING FIBERGLASS STRIPED 6FT   021954        219.54
```

This display appears if the ORDHDRP file does not contain a record for the order number entered on the first display:

```
Existing Order Inquiry                    Total  00000000
                                          Status
Order 12400                               Open
Date 000000                               Customer order
Cust #                                    Ship via
                             00000        Printed date 000000
                           Invoice  00000                    Mth 00  Year 00
Item  Qty   Item description                Price    Extension









Order number not found
```

Figure 37 on page 167 shows an example payment update program, ARC010, with the related DDS and example display screens. For the DDS for the customer master file, CUSMSTP, refer to Figure 35 on page 146.

In this example, payments from customers are registered. The clerk is prompted to enter one or more customer numbers and the amount of money to be credited to each customer's account. The program checks the customer number and unconditionally accepts any payment for an existing customer who has invoices outstanding. If an overpayment will result from the amount of the payment from a customer, the clerk is given the option to accept or reject the payment. If no customer record exists for a customer number, an error message is issued. Payments can be entered until the clerk ends the program by pressing function key 12.



IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

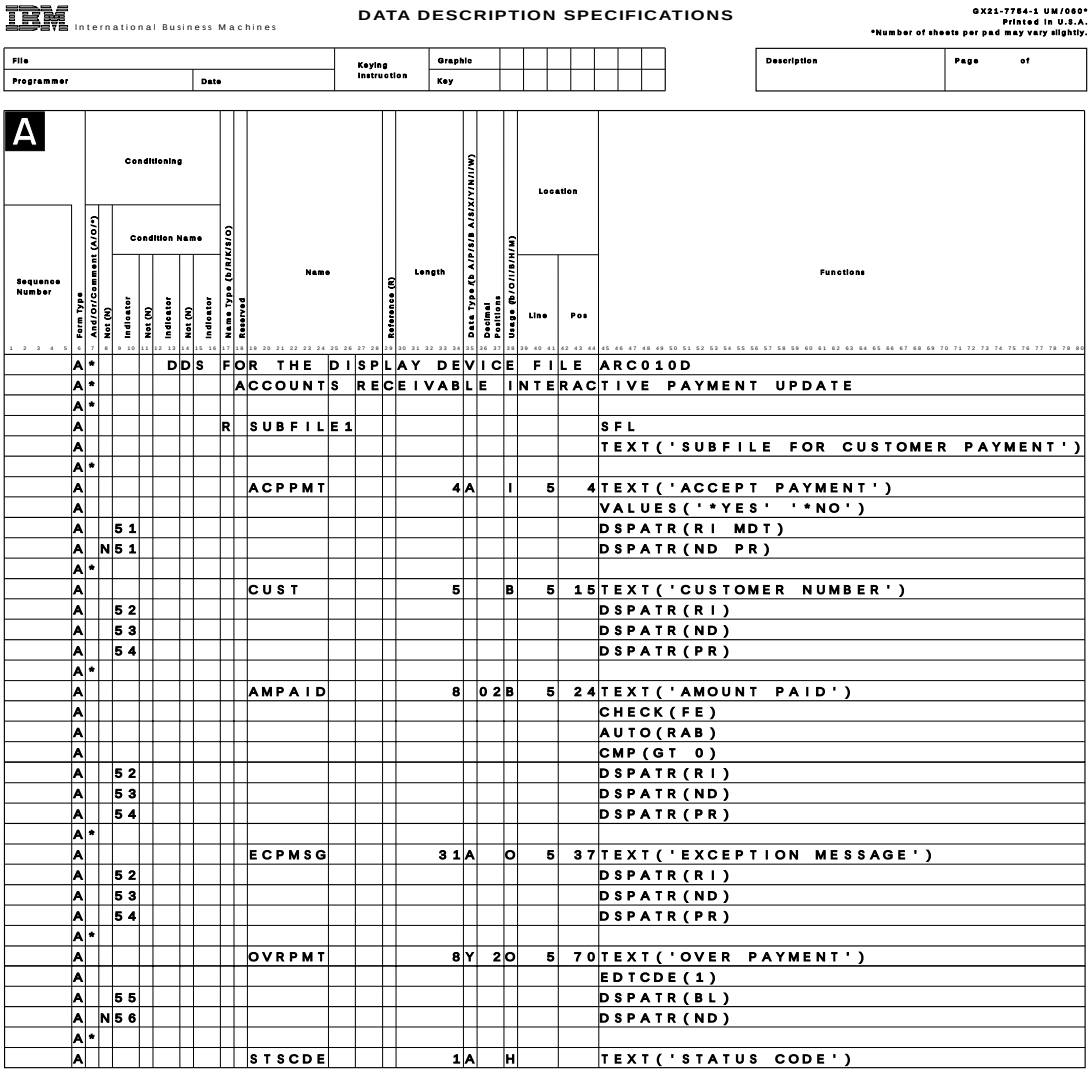
GX21-7754-1 UM/080*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Not (N)	Conditioning: Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (b/R/K/S/C)	Reserved	Name	Reference (R)	Length	Data Type (b/P/S/B A/S/X/Y/N/I/M)	Decimal Positions	Usage (b/O/I/B/H/M)	Location: Line	Location: Pos	Functions
	A	*	*	LOGICAL							ORDHDR	L		ORDER	FILE				OF ORDHDR
	A								R		ORDHDR								PFILE(ORDHDRP)
	A	*																	
	A										CUST								
	A										INVNUM								
	A										ORDER								
	A										ORDDAT								
	A										CUSORD								
	A										SHPVIA								
	A										ORDSTS								
	A										OPRNAM								
	A										ORDAMT								
	A										CUSTYP								
	A										PRTDAT								
	A										OPNSTS								
	A										TOTLIN								
	A										ACTMTH								
	A										ACTYR								
	A										STATE								
	A										AMPAID								
	A								K		CUST								
	A								K		INYNUM								
	A																		

Figure 37 (Part 1 of 14). Example Payment Update Program



IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

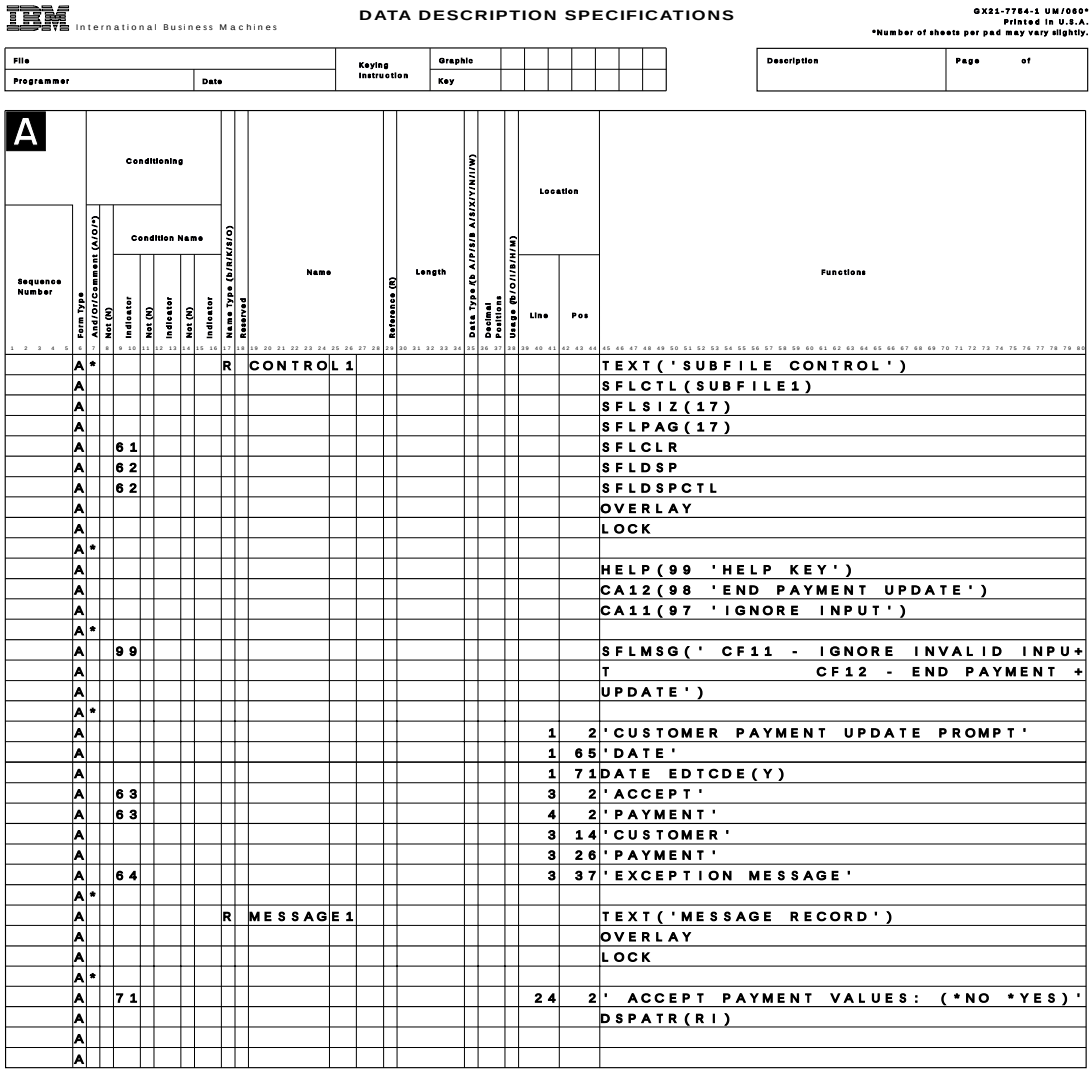
GX21-7784-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

A

Sequence Number	Form Type	Conditioning: And/Or/Comment (A/O/C)	Conditioning: Not (N)	Conditioning: Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (b/H/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (b/A/P/S/B/A1/B1/X1/Y1/N1/I1/W)	Decimal Positions	Usage (b/I/O/I/B/H/M)	Location: Line	Location: Pos	Functions
	A	*																	
	A	*				DDS	FOR	THE	DISP	LAY	DEVICE	FILE							ARC010D
	A	*				ACCOUNTS	REC	EIVABL	E		INTERAC	TIVE	PAYMENT	UPDATE					
	A	*																	
	A								R		SUBFILE1								SFL
	A																		TEXT('SUBFILE FOR CUSTOMER PAYMENT')
	A	*																	
	A										ACPPMT		4	A		I	5	4	TEXT('ACCEPT PAYMENT')
	A																		VALUES('*YES' '*NO')
	A			51															DSPATR(RI MDT)
	A		N	51															DSPATR(ND PR)
	A	*																	
	A										CUST		5		B	5	15		TEXT('CUSTOMER NUMBER')
	A			52															DSPATR(RI)
	A			53															DSPATR(ND)
	A			54															DSPATR(PR)
	A	*																	
	A										AMPAID		8	02	B	5	24		TEXT('AMOUNT PAID')
	A																		CHECK(FE)
	A																		AUTO(RAB)
	A																		CMP(GT 0)
	A			52															DSPATR(RI)
	A			53															DSPATR(ND)
	A			54															DSPATR(PR)
	A	*																	
	A										ECPMSG		31	A		O	5	37	TEXT('EXCEPTION MESSAGE')
	A			52															DSPATR(RI)
	A			53															DSPATR(ND)
	A			54															DSPATR(PR)
	A	*																	
	A										OVRPMT		8	Y	2	O	5	70	TEXT('OVER PAYMENT')
	A																		EDTCDE(1)
	A			55															DSPATR(BL)
	A		N	56															DSPATR(ND)
	A	*																	
	A										STSCDE		1	A		H			TEXT('STATUS CODE')

Figure 37 (Part 2 of 14). Example Payment Update Program



IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7764-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								
Programmer	Date		Key								

Description	Page of

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Name Type (b/R/K/F/S/O)	Reserved	Name	Reference (R)	Length	Data Type (b A/P/S/B A/S/X/Y/N/I/1/W)	Decimal Positions	Usage (B/O/I/H/M)	Location: Line	Location: Pos	Functions
	A	*							R		CONTROL1								TEXT('SUBFILE CONTROL')
	A																		SFLCTL(SUBFILE1)
	A																		SFLSIZ(17)
	A																		SFLPAG(17)
	A			61															SFLCLR
	A			62															SFLDSP
	A			62															SFLDSPCTL
	A																		OVERLAY
	A																		LOCK
	A	*																	
	A																		HELP(99 'HELP KEY')
	A																		CA12(98 'END PAYMENT UPDATE')
	A																		CA11(97 'IGNORE INPUT')
	A	*																	
	A			99															SFLMSG(' CF11 - IGNORE INVALID INPU+
	A																		T CF12 - END PAYMENT +
	A																		UPDATE')
	A	*																	
	A																1	2	'CUSTOMER PAYMENT UPDATE PROMPT'
	A																1	65	'DATE'
	A																1	71	DATE EDTCDE(Y)
	A			63													3	2	'ACCEPT'
	A			63													4	2	'PAYMENT'
	A																3	14	'CUSTOMER'
	A																3	26	'PAYMENT'
	A			64													3	37	'EXCEPTION MESSAGE'
	A	*																	
	A								R		MESSAGE1								TEXT('MESSAGE RECORD')
	A																		OVERLAY
	A																		LOCK
	A	*																	
	A			71													24	2	' ACCEPT PAYMENT VALUES: (*NO *YES)'
	A																		DSPATR(RI)
	A																		
	A																		

Figure 37 (Part 3 of 14). Example Payment Update Program

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.  ARC010.
    3  000300 ENVIRONMENT DIVISION.
    4  000400 CONFIGURATION SECTION.
    5  000500 SOURCE-COMPUTER. IBM-S38.
    6  000600 OBJECT-COMPUTER. IBM-S38.
    7  000700 INPUT-OUTPUT SECTION.
    8  000800 FILE-CONTROL.
    9  000900     SELECT CUSTOMER-INVOICE-FILE
   10  001000         ASSIGN TO DATABASE-ORDHDRL
   11  001100         ORGANIZATION IS INDEXED
   12  001200         ACCESS MODE IS SEQUENTIAL
   13  001300         RECORD KEY IS COMP-KEY
   14  001400         FILE STATUS IS STATUS-CODE-ONE.
   15  001500     SELECT CUSTOMER-MASTER-FILE
   16  001600         ASSIGN TO DATABASE-CUSMSTP
   17  001700         ORGANIZATION IS INDEXED
   18  001800         ACCESS IS RANDOM
   19  001900         RECORD KEY IS CUST OF CUSTOMER-MASTER-RECORD.
   20  002000     SELECT PAYMENT-UPDATE-DISPLAY-FILE
   21  002100         ASSIGN TO WORKSTATION-ARC010D
   22  002200         ORGANIZATION IS TRANSACTION
   23  002300         ACCESS IS DYNAMIC
   24  002400         RELATIVE KEY IS REL-NUMBER
   25  002500         FILE STATUS IS STATUS-CODE-ONE
   26  002600         CONTROL-AREA IS WS-CONTROL.
   27  002700
   28  002800 DATA DIVISION.
   29  002900 FILE SECTION.
   30  003000 FD  CUSTOMER-INVOICE-FILE
   31  003100     LABEL RECORDS ARE STANDARD.
   32  003200 01  CUSTOMER-INVOICE-RECORD.
   33  003300     COPY DDS-ORDHDR OF ORDHDRL.
      +000001*    I-O FORMAT:ORDHDR     FROM FILE ORDHDRL    OF LIBRARY COB38EX           ORDHDR
      +000002*                                                                            ORDHDR
      +000003*THE KEY DEFINITIONS FOR RECORD FORMAT  ORDHDR                               ORDHDR
      +000004*  NUMBER              NAME              RETRIEVAL     TYPE    ALTSEQ       ORDHDR
      +000005*   0001   CUST                          ASCENDING      AN      NO          ORDHDR
      +000006*   0002   INVNUM                        ASCENDING   SIGNED     NO          ORDHDR
   34 +000007       05  ORDHDR.                                                          ORDHDR
   35 +000008           06 CUST             PIC X(5).                                    ORDHDR
      +000009*                           CUSTOMER NUMBER                                ORDHDR
   36 +000010           06 INVNUM           PIC S9(5)        COMP-3.                     ORDHDR
      +000011*                           INVOICE NUMBER                                 ORDHDR
   37 +000012           06 ORDERN           PIC S9(5)        COMP-3.                     ORDHDR
      +000013*                           ORDER NUMBER                                   ORDHDR
   38 +000014           06 ORDDAT           PIC S9(6)        COMP-3.                     ORDHDR
      +000015*                           DATE ORDER ENTERED                             ORDHDR
   39 +000016           06 CUSORD           PIC X(15).                                   ORDHDR
      +000017*                           CUSTOMER PURCHASE ORDER NUMBER                 ORDHDR
   40 +000018           06 SHPVIA           PIC X(15).                                   ORDHDR
      +000019*                           SHIPPING INSTRUCTIONS                          ORDHDR
   41 +000020           06 ORDSTS           PIC S9(1)        COMP-3.                     ORDHDR
      +000021*                           ORDER STATUS 1PCS 2CNT  3CHK 4RDY 5PRT 6PC     ORDHDR
   42 +000022           06 OPRNAM           PIC X(10).                                   ORDHDR
```

Figure 37 (Part 4 of 14). Example Payment Update Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      +000023*                        OPERATOR WHO ENTERED ORD                            ORDHDR
   43 +000024           06 ORDAMT            PIC S9(6)V9(2)   COMP-3.                     ORDHDR
      +000025*                        DOLLAR AMOUNT OF ORDER                              ORDHDR
   44 +000026           06 CUSTYP            PIC S9(1)        COMP-3.                     ORDHDR
      +000027*                        CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT          ORDHDR
   45 +000028           06 PRTDAT            PIC S9(6)        COMP-3.                     ORDHDR
      +000029*                        DATE ORDER WAS PRINTED                              ORDHDR
   46 +000030           06 OPNSTS            PIC S9(1)        COMP-3.                     ORDHDR
      +000031*                        ORDER OPEN STATUS 1=OPEN 2= CLOSE 3=CANCEL          ORDHDR
   47 +000032           06 TOTLIN            PIC S9(3)        COMP-3.                     ORDHDR
      +000033*                        TOTAL LINE ITEMS IN ORDER                           ORDHDR
   48 +000034           06 ACTMTH            PIC S9(2)        COMP-3.                     ORDHDR
      +000035*                        ACCOUNTING MONTH OF SALE                            ORDHDR
   49 +000036           06 ACTYR             PIC S9(2)        COMP-3.                     ORDHDR
      +000037*                        ACCOUNTING YEAR OF SALE                             ORDHDR
   50 +000038           06 STATE             PIC X(2).                                    ORDHDR
      +000039*                        STATE                                               ORDHDR
   51 +000040           06 AMPAID            PIC S9(6)V9(2)   COMP-3.                     ORDHDR
      +000041*                        AMOUNT PAID                                         ORDHDR
   52  003400 66  COMP-KEY RENAMES CUST THRU INVNUM.
   53  003500
   54  003600 FD  CUSTOMER-MASTER-FILE
   55  003700     LABEL RECORDS ARE STANDARD.
   56  003800 01  CUSTOMER-MASTER-RECORD.
   57  003900     COPY DDS-CUSMST OF CUSMSTP.
      +000001*    I-O FORMAT:CUSMST     FROM FILE CUSMSTP    OF LIBRARY COB38EX           CUSMST
      +000002*                        ORDER HEADER RECORD                                 CUSMST
      +000003*THE KEY DEFINITIONS FOR RECORD FORMAT  CUSMST                                CUSMST
      +000004*  NUMBER              NAME               RETRIEVAL     TYPE    ALTSEQ       CUSMST
      +000005*   0001   CUST                          ASCENDING      AN      NO          CUSMST
   58 +000006       05  CUSMST.                                                          CUSMST
   59 +000007           06 CUST              PIC X(5).                                    CUSMST
      +000008*                        CUSTOMER NUMBER                                     CUSMST
   60 +000009           06 NAME              PIC X(25).                                   CUSMST
      +000010*                        CUSTOMER NAME                                       CUSMST
   61 +000011           06 ADDR              PIC X(20).                                   CUSMST
      +000012*                        CUSTOMER ADDRESS                                    CUSMST
   62 +000013           06 CITY              PIC X(20).                                   CUSMST
      +000014*                        CUSTOMER CITY                                       CUSMST
   63 +000015           06 STATE             PIC X(2).                                    CUSMST
      +000016*                        STATE                                               CUSMST
   64 +000017           06 ZIP               PIC S9(5)        COMP-3.                     CUSMST
      +000018*                        ZIP CODE                                            CUSMST
   65 +000019           06 SRHCOD            PIC X(6).                                    CUSMST
      +000020*                        CUSTOMER NUMBER SEARCH CODE                         CUSMST
   66 +000021           06 CUSTYP            PIC S9(1)        COMP-3.                     CUSMST
      +000022*                        CUSTOMER TYPE 1=GOV 2=SCH 3=BUS 4=PVT 5=OT          CUSMST
   67 +000023           06 ARBAL             PIC S9(6)V9(2)   COMP-3.                     CUSMST
      +000024*                        ACCOUNTS REC. BALANCE                               CUSMST
   68 +000025           06 ORDBAL            PIC S9(6)V9(2)   COMP-3.                     CUSMST
      +000026*                        A/R AMT. IN ORDER FILE                              CUSMST
   69 +000027           06 LSTAMT            PIC S9(6)V9(2)   COMP-3.                     CUSMST
      +000028*                        LAST AMT. PAID IN A/R                               CUSMST
   70 +000029           06 LSTDAT            PIC S9(6)        COMP-3.                     CUSMST
      +000030*                        LAST DATE PAID IN A/R                               CUSMST
```

Figure 37 (Part 5 of 14). Example Payment Update Program

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   71 +000031           06 CRDLMT            PIC S9(6)V9(2)   COMP-3.                      CUSMST
      +000032*                             CUSTOMER CREDIT LIMIT                           CUSMST
   72 +000033           06 SLSYR             PIC S9(8)V9(2)   COMP-3.                      CUSMST
      +000034*                             CUSTOMER SALES THIS YEAR                        CUSMST
   73 +000035           06 SLSLYR            PIC S9(8)V9(2)   COMP-3.                      CUSMST
      +000036*                             CUSTOMER SALES LAST YEAR                        CUSMST
   74  004000
   75  004100 FD  PAYMENT-UPDATE-DISPLAY-FILE
   76  004200     LABEL RECORDS ARE OMITTED.
   77  004300 01  PAYMENT-UPDATE-DISPLAY-RECORD.
   78  004400     COPY DDS-ALL-FORMATS OF ARC010D.
   79 +000001       05  ARC010D-RECORD PIC X(59).                                          <-ALL-FMTS
      +000002*  INPUT FORMAT:SUBFILE1   FROM FILE ARC010D    OF LIBRARY COB38EX           <-ALL-FMTS
      +000003*                          SUBFILE FOR CUSTOMER PAYMENT                      <-ALL-FMTS
   80 +000004       05  SUBFILE1-I    REDEFINES ARC010D-RECORD.                          <-ALL-FMTS
   81 +000005           06 ACPPMT            PIC X(4).                                   <-ALL-FMTS
      +000006*                             ACCEPT PAYMENT                                  <-ALL-FMTS
   82 +000007           06 CUST              PIC X(5).                                   <-ALL-FMTS
      +000008*                             CUSTOMER NUMBER                                 <-ALL-FMTS
   83 +000009           06 AMPAID            PIC S9(6)V9(2).                             <-ALL-FMTS
      +000010*                             AMOUNT PAID                                     <-ALL-FMTS
   84 +000011           06 ECPMSG            PIC X(31).                                  <-ALL-FMTS
      +000012*                             EXCEPTION MESSAGE                               <-ALL-FMTS
   85 +000013           06 OVRPMT            PIC S9(6)V9(2).                             <-ALL-FMTS
      +000014*                             OVERPAYMENT                                     <-ALL-FMTS
   86 +000015           06 STSCDE            PIC X(1).                                   <-ALL-FMTS
      +000016*                             STATUS CODE                                     <-ALL-FMTS
      +000017* OUTPUT FORMAT:SUBFILE1   FROM FILE ARC010D    OF LIBRARY COB38EX           <-ALL-FMTS
      +000018*                          SUBFILE FOR CUSTOMER PAYMENT                      <-ALL-FMTS
   87 +000019       05  SUBFILE1-O    REDEFINES ARC010D-RECORD.                          <-ALL-FMTS
   88 +000020           06 SUBFILE1-O-INDIC.                                             <-ALL-FMTS
   89 +000021                07 IN51         PIC 1  INDIC 51.                           <-ALL-FMTS
   90 +000022                07 IN52         PIC 1  INDIC 52.                           <-ALL-FMTS
   91 +000023                07 IN53         PIC 1  INDIC 53.                           <-ALL-FMTS
   92 +000024                07 IN54         PIC 1  INDIC 54.                           <-ALL-FMTS
   93 +000025                07 IN55         PIC 1  INDIC 55.                           <-ALL-FMTS
   94 +000026                07 IN56         PIC 1  INDIC 56.                           <-ALL-FMTS
   95 +000027           06 CUST              PIC X(5).                                   <-ALL-FMTS
      +000028*                             CUSTOMER NUMBER                                 <-ALL-FMTS
   96 +000029           06 AMPAID            PIC S9(6)V9(2).                             <-ALL-FMTS
      +000030*                             AMOUNT PAID                                     <-ALL-FMTS
   97 +000031           06 ECPMSG            PIC X(31).                                  <-ALL-FMTS
      +000032*                             EXCEPTION MESSAGE                               <-ALL-FMTS
   98 +000033           06 OVRPMT            PIC S9(6)V9(2).                             <-ALL-FMTS
      +000034*                             OVERPAYMENT                                     <-ALL-FMTS
   99 +000035           06 STSCDE            PIC X(1).                                   <-ALL-FMTS
      +000036*                             STATUS CODE                                     <-ALL-FMTS
      +000037*  INPUT FORMAT:CONTROL1   FROM FILE ARC010D    OF LIBRARY COB38EX           <-ALL-FMTS
      +000038*                          SUBFILE CONTROL                                   <-ALL-FMTS
  100 +000039       05  CONTROL1-I    REDEFINES ARC010D-RECORD.                          <-ALL-FMTS
  101 +000040           06 CONTROL1-I-INDIC.                                             <-ALL-FMTS
  102 +000041                07 IN99         PIC 1  INDIC 99.                           <-ALL-FMTS
      +000042*                             HELP KEY                                        <-ALL-FMTS
  103 +000043                07 IN98         PIC 1  INDIC 98.                           <-ALL-FMTS
      +000044*                             END PAYMENT UPDATE                              <-ALL-FMTS
```

Figure 37 (Part 6 of 14). Example Payment Update Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  104 +000045               07 IN97         PIC 1  INDIC 97.                            <-ALL-FMTS
      +000046*                             IGNORE INPUT                                  <-ALL-FMTS
      +000047* OUTPUT FORMAT:CONTROL1   FROM FILE ARC010D    OF LIBRARY COB38EX           <-ALL-FMTS
      +000048*                             SUBFILE CONTROL                               <-ALL-FMTS
  105 +000049       05  CONTROL1-O    REDEFINES ARC010D-RECORD.                          <-ALL-FMTS
  106 +000050           06 CONTROL1-O-INDIC.                                             <-ALL-FMTS
  107 +000051               07 IN61         PIC 1  INDIC 61.                            <-ALL-FMTS
  108 +000052               07 IN62         PIC 1  INDIC 62.                            <-ALL-FMTS
  109 +000053               07 IN99         PIC 1  INDIC 99.                            <-ALL-FMTS
      +000054*                             HELP KEY                                      <-ALL-FMTS
  110 +000055               07 IN63         PIC 1  INDIC 63.                            <-ALL-FMTS
  111 +000056               07 IN64         PIC 1  INDIC 64.                            <-ALL-FMTS
      +000057*  INPUT FORMAT:MESSAGE1   FROM FILE ARC010D    OF LIBRARY COB38EX           <-ALL-FMTS
      +000058*                             MESSAGE RECORD                                <-ALL-FMTS
      +000059*      05  MESSAGE1-I    REDEFINES ARC010D-RECORD.                          <-ALL-FMTS
      +000060* OUTPUT FORMAT:MESSAGE1   FROM FILE ARC010D    OF LIBRARY COB38EX           <-ALL-FMTS
      +000061*                             MESSAGE RECORD                                <-ALL-FMTS
  112 +000062       05  MESSAGE1-O    REDEFINES ARC010D-RECORD.                          <-ALL-FMTS
  113 +000063           06 MESSAGE1-O-INDIC.                                             <-ALL-FMTS
  114 +000064               07 IN71         PIC 1  INDIC 71.                            <-ALL-FMTS
  115  004500
  116  004600 WORKING-STORAGE SECTION.
  117  004700
  118  004800 01  REL-NUMBER                  PIC 9(05)
  119  004900                                    VALUE ZEROS.
  120  005000
  121  005100 01  WS-CONTROL.
  122  005200     05  WS-IND                   PIC X(02).
  123  005300     05  WS-FORMAT                PIC X(10).
  124  005400 01  SYSTEM-DATE.
  125  005500     05  SYSTEM-YEAR              PIC 99.
  126  005600     05  SYSTEM-MONTH             PIC 99.
  127  005700     05  SYSTEM-DAY               PIC 99.
  128  005800 01  PROGRAM-DATE.
  129  005900     05  PROGRAM-MONTH            PIC 99.
  130  006000     05  PROGRAM-DAY              PIC 99.
  131  006100     05  PROGRAM-YEAR             PIC 99.
  132  006200 01  FILE-DATE REDEFINES PROGRAM-DATE
  133  006300                                  PIC S9(6).
  134  006400 01  EXCEPTION-STATUS.
  135  006500     05  STATUS-CODE-ONE          PIC XX.
  136  006600         88  SUBFILE-IS-FULL      VALUE '9M'.
  137  006700 01  EXCEPTION-MESSAGES.
  138  006800     05  MESSAGE-ONE              PIC X(31)
  139  006900         VALUE 'CUSTOMER DOES NOT EXIST        '.
  140  007000     05  MESSAGE-TWO              PIC X(31)
  141  007100         VALUE 'NO INVOICES EXIST FOR CUSTOMER '.
  142  007200     05  MESSAGE-THREE            PIC X(31)
  143  007300         VALUE 'CUSTOMER HAS AN OVER PAYMENT OF'.
  144  007400 01  PROGRAM-VARIABLES.
  145  007500     05  AMOUNT-OWED              PIC S9(6)V99.
  146  007600     05  AMOUNT-PAID              PIC S9(6)V99.
  147  007700     05  INVOICE-BALANCE          PIC S9(6)V99.
  148  007800 01  ERRPGM-PARAMETERS.
  149  007900     05  DISPLAY-PARAMETER        PIC X(8)
```

Figure 37 (Part 7 of 14). Example Payment Update Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  150  008000                                    VALUE 'ARC010D'.
  151  008100    05  DUMMY-ONE                 PIC X(6)
  152  008200                                    VALUE SPACES.
  153  008300    05  DUMMY-TWO                 PIC X(6)
  154  008400                                    VALUE SPACES.
  155  008500    05  STATUS-CODE-TWO.
  156  008600       10  PRIMARY                PIC X(1).
  157  008700       10  SECONDARY              PIC X(1).
  158  008800       10  FILLER                 PIC X(5)
  159  008900                                    VALUE SPACES.
  160  009000    05  DUMMY-THREE               PIC X(10)
  161  009100                                    VALUE SPACES.
  162  009200
  163  009300 01  SWITCH-AREA.
  164  009400    05  SW01                      PIC 1.
  165  009500       88  WRITE-DISPLAY                VALUE B'1'.
  166  009600       88  READ-DISPLAY                 VALUE B'0'.
  167  009700    05  SW02                      PIC 1.
  168  009800       88  SUBFILE1-FORMAT              VALUE B'1'.
  169  009900       88  NOT-SUBFILE1-FORMAT          VALUE B'0'.
  170  010000    05  SW03                      PIC 1.
  171  010100       88  CONTROL1-FORMAT              VALUE B'1'.
  172  010200       88  NOT-CONTROL1-FORMAT          VALUE B'1'.
  173  010300    05  SW04                      PIC 1.
  174  010400       88  NO-MORE-TRANSACTIONS-EXIST   VALUE B'1'.
  175  010500       88  TRANSACTIONS-EXIST           VALUE B'0'.
  176  010600    05  SW05                      PIC 1.
  177  010700       88  CUSTOMER-NOT-FOUND           VALUE B'1'.
  178  010800       88  CUSTOMER-EXIST               VALUE B'0'.
  179  010900    05  SW06                      PIC 1.
  180  011000       88  NO-MORE-INVOICES-EXIST       VALUE B'1'.
  181  011100       88  CUSTOMER-INVOICE-EXIST       VALUE B'0'.
  182  011200    05  SW07                      PIC 1.
  183  011300       88  NO-MORE-PAYMENT-EXIST        VALUE B'0'.
  184  011400       88  PAYMENT-EXIST                VALUE B'0'.
  185  011500    05  SW08                      PIC 1.
  186  011600       88  INPUT-ERRORS-EXIST           VALUE B'1'.
  187  011700       88  NO-INPUT-ERRORS-EXIST        VALUE B'0'.
  188  011800    05  SW09                      PIC 1.
  189  011900       88  OVER-PAYMENT-DISPLAYED-ONCE  VALUE B'1'.
  190  012000       88  OVER-PAYMENT-NOT-DISPLAYED   VALUE B'0'.
  191  012100
  192  012200 01  INDICATOR-AREA.
  193  012300    05  IN99                      PIC 1 INDIC 99.
  194  012400       88 HELP-IS-NEEDED              VALUE B'1'.
  195  012500       88 HELP-IS-NOT-NEEDED          VALUE B'0'.
  196  012600    05  IN98                      PIC 1 INDIC 98.
  197  012700       88 END-OF-PAYMENT-UPDATE       VALUE B'1'.
  198  012800    05  IN97                      PIC 1 INDIC 97.
  199  012900       88 IGNORE-INPUT                VALUE B'1'.
  200  013000    05  IN51                      PIC 1 INDIC 51.
  201  013100       88 DISPLAY-ACCEPT-PAYMENT      VALUE B'1'.
  202  013200       88 DO-NOT-DISPLAY-ACCEPT-PAYMENT  VALUE B'0'.
  203  013300    05  IN52                      PIC 1 INDIC 52.
  204  013400       88 REVERSE-FIELD-IMAGE         VALUE B'1'.
```

Figure 37 (Part 8 of 14). Example Payment Update Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  205  013500       88 DO-NOT-REVERSE-FIELD-IMAGE     VALUE B'0'.
  206  013600    05  IN53                        PIC 1 INDIC 53.
  207  013700       88 DO-NOT-DISPLAY-FIELD           VALUE B'1'.
  208  013800       88 DISPLAY-FIELD                  VALUE B'0'.
  209  013900    05  IN54                        PIC 1 INDIC 54.
  210  014000       88 PROTECT-INPUT-FIELD            VALUE B'1'.
  211  014100       88 DO-NOT-PROTECT-INPUT-FIELD     VALUE B'0'.
  212  014200    05  IN55                        PIC 1 INDIC 55.
  213  014300       88 MAKE-FIELD-BLINK               VALUE B'1'.
  214  014400       88 DO-NOT-MAKE-FIELD-BLINK        VALUE B'0'.
  215  014500    05  IN56                        PIC 1 INDIC 56.
  216  014600       88 DISPLAY-OVER-PAYMENT           VALUE B'1'.
  217  014700       88 DO-NOT-DISPLAY-OVER-PAYMENT    VALUE B'0'.
  218  014800    05  IN61                        PIC 1 INDIC 61.
  219  014900       88 CLEAR-SUBFILE                  VALUE B'1'.
  220  015000       88 DO-NOT-CLEAR-SUBFILE           VALUE B'0'.
  221  015100    05  IN62                        PIC 1 INDIC 62.
  222  015200       88 DISPLAY-SCREEN                 VALUE B'1'.
  223  015300       88 DO-NOT-DISPLAY-SCREEN          VALUE B'0'.
  224  015400    05  IN63                        PIC 1 INDIC 63.
  225  015500       88 DISPLAY-ACCEPT-HEADING         VALUE B'1'.
  226  015600       88 DO-NOT-DISPLAY-ACCEPT-HEADING  VALUE B'0'.
  227  015700    05  IN64                        PIC 1 INDIC 64.
  228  015800       88 DISPLAY-EXCEPTION              VALUE B'1'.
  229  015900       88 DO-NOT-DISPLAY-EXCEPTION       VALUE B'0'.
  230  016000    05  IN71                        PIC 1 INDIC 71.
  231  016100       88 DISPLAY-ACCEPT-MESSAGE         VALUE B'1'.
  232  016200       88 DO-NOT-DISPLAY-ACCEPT-MESSAGE  VALUE B'0'.
  233  016300
  234  016400 PROCEDURE DIVISION.
       016500
       016600 DECLARATIVES.
       016700
       016800 TRANSACTION-ERROR SECTION.
       016900     USE AFTER STANDARD ERROR PROCEDURE
       017000         PAYMENT-UPDATE-DISPLAY-FILE.
       017100 WORK-STATION-ERROR-HANDLER.
  235  017200     IF SUBFILE-IS-FULL THEN
       017300       NEXT SENTENCE
       017400     ELSE
  236  017500       DISPLAY 'ERROR IN PAYMENT-UPDATE' STATUS-CODE-ONE.
       017600 END DECLARATIVES.
       017700
       017800 CUSTOMER-PAYMENT-UPDATE SECTION.
       017900 MAINLINE-ROUTINE.
  237  018000     PERFORM SET-UP-ROUTINE.
  238  018100     PERFORM PROCESS-TRANSACTION-FILE
       018200       UNTIL END-OF-PAYMENT-UPDATE.
  239  018300     PERFORM CLEAN-UP-ROUTINE.
       018400
       018500 SET-UP-ROUTINE.
  240  018600     OPEN I-O      CUSTOMER-INVOICE-FILE
       018700                   CUSTOMER-MASTER-FILE
       018800                   PAYMENT-UPDATE-DISPLAY-FILE.
  241  018900     MOVE ALL B'0' TO INDICATOR-AREA
```

Figure 37 (Part 9 of 14). Example Payment Update Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      019000                   SWITCH-AREA.
  242 019100     ACCEPT SYSTEM-DATE FROM DATE.
  243 019200     MOVE SYSTEM-YEAR  TO PROGRAM-YEAR.
  244 019300     MOVE SYSTEM-MONTH TO PROGRAM-MONTH.
  245 019400     MOVE SYSTEM-DATE  TO PROGRAM-DAY.
  246 019500     SET WRITE-DISPLAY
      019600         CONTROL1-FORMAT
      019700         DO-NOT-DISPLAY-OVER-PAYMENT
      019800         DO-NOT-PROTECT-INPUT-FIELD
      019900         DO-NOT-REVERSE-FIELD-IMAGE
      020000         DO-NOT-MAKE-FIELD-BLINK
      020100         CLEAR-SUBFILE TO TRUE.
  247 020200     MOVE CORR INDICATOR-AREA TO CONTROL1-O-INDIC.
  248 020300     WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      020400         FORMAT IS 'CONTROL1'.
  249 020500     SET DO-NOT-CLEAR-SUBFILE TO TRUE.
  250 020600     PERFORM INITIALIZE-SUBFILE-RECORD 17 TIMES.
  251 020700     SET DISPLAY-SCREEN TO TRUE.
  252 020800     MOVE CORR INDICATOR-AREA TO CONTROL1-O-INDIC.
  253 020900     WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      021000         FORMAT IS 'CONTROL1'.
  254 021100     READ PAYMENT-UPDATE-DISPLAY-FILE RECORD
      021200         FORMAT IS 'CONTROL1'.
  255 021300     MOVE CORR CONTROL1-I-INDIC TO INDICATOR-AREA.
      021400 PROCESS-TRANSACTION-FILE.
  256 021500     IF HELP-IS-NOT-NEEDED THEN
  257 021600        IF  IGNORE-INPUT THEN
  258 021700            SET WRITE-DISPLAY
      021800                CONTROL1-FORMAT
      021900                CLEAR-SUBFILE
      022000                DISPLAY-FIELD
      022100                DO-NOT-DISPLAY-OVER-PAYMENT
      022200                DO-NOT-PROTECT-INPUT-FIELD
      022300                DO-NOT-REVERSE-FIELD-IMAGE
      022400                DO-NOT-DISPLAY-ACCEPT-PAYMENT
      022500                DO-NOT-DISPLAY-ACCEPT-HEADING
      022600                DO-NOT-DISPLAY-ACCEPT-MESSAGE
      022700                DO-NOT-MAKE-FIELD-BLINK TO TRUE
  259 022800            MOVE CORR INDICATOR-AREA TO CONTROL1-O-INDIC
  260 022900            WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      023000                FORMAT IS 'CONTROL1'
  261 023100            SET DO-NOT-CLEAR-SUBFILE TO TRUE
  262 023200            MOVE 0 TO REL-NUMBER
  263 023300            PERFORM INITIALIZE-SUBFILE-RECORD 17 TIMES
      023400         ELSE
  264 023500            SET TRANSACTIONS-EXIST
      023600                DO-NOT-DISPLAY-ACCEPT-HEADING
      023700                DO-NOT-DISPLAY-ACCEPT-MESSAGE
      023800                DO-NOT-DISPLAY-EXCEPTION TO TRUE
  265 023900            PERFORM READ-MODIFIED-SUBFILE-RECORD
  266 024000            PERFORM TRANSACTION-VALIDATION
      024100                UNTIL NO-MORE-TRANSACTIONS-EXIST
  267 024200            SET NO-INPUT-ERRORS-EXIST TO TRUE
  268 024300            PERFORM TEST-FOR-RECORD-INPUT-ERRORS
      024400                VARYING REL-NUMBER
```

Figure 37 (Part 10 of 14). Example Payment Update Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      024500                FROM    1
      024600                BY      1
      024700                UNTIL   REL-NUMBER IS GREATER THAN 17
      024800                OR      INPUT-ERRORS-EXIST
  269 024900            IF  NO-INPUT-ERRORS-EXIST THEN
  270 025000                IF  OVER-PAYMENT-DISPLAYED-ONCE THEN
  271 025100                    SET WRITE-DISPLAY
      025200                        CONTROL1-FORMAT
      025300                        DO-NOT-DISPLAY-OVER-PAYMENT
      025400                        DO-NOT-PROTECT-INPUT-FIELD
      025500                        DO-NOT-REVERSE-FIELD-IMAGE
      025600                        DO-NOT-MAKE-FIELD-BLINK
      025700                        DO-NOT-DISPLAY-ACCEPT-PAYMENT
      025800                        DO-NOT-DISPLAY-ACCEPT-HEADING
      025900                        DO-NOT-DISPLAY-ACCEPT-MESSAGE
      026000                        DO-NOT-DISPLAY-EXCEPTION
      026100                        CLEAR-SUBFILE
      026200                        DISPLAY-FIELD
      026300                          TO TRUE
  272 026400                    MOVE CORR INDICATOR-AREA TO CONTROL1-O-INDIC
  273 026500                    WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      026600                        FORMAT IS 'CONTROL1'
  274 026700                    SET DO-NOT-CLEAR-SUBFILE TO TRUE
  275 026800                    MOVE 0 TO REL-NUMBER
  276 026900                    PERFORM INITIALIZE-SUBFILE-RECORD 17 TIMES
      027000                ELSE
  277 027100                    SET OVER-PAYMENT-DISPLAYED-ONCE TO TRUE
      027200            ELSE
      027300                NEXT SENTENCE
      027400     ELSE
      027500         NEXT SENTENCE.
  278 027600     SET WRITE-DISPLAY, DISPLAY-SCREEN TO TRUE.
  279 027700     MOVE CORR INDICATOR-AREA TO MESSAGE1-O-INDIC.
  280 027800     WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      027900         FORMAT IS 'MESSAGE1'.
  281 028000     SET WRITE-DISPLAY, CONTROL1-FORMAT TO TRUE.
  282 028100     MOVE CORR INDICATOR-AREA TO CONTROL1-O-INDIC.
  283 028200     WRITE PAYMENT-UPDATE-DISPLAY-RECORD
      028300         FORMAT IS 'CONTROL1'.
  284 028400     READ PAYMENT-UPDATE-DISPLAY-FILE RECORD
      028500         FORMAT IS 'CONTROL1'.
  285 028600     MOVE CORR CONTROL1-I-INDIC TO INDICATOR-AREA.
      028700 READ-MODIFIED-SUBFILE-RECORD.
  286 028800     READ SUBFILE PAYMENT-UPDATE-DISPLAY-FILE
      028900         NEXT MODIFIED RECORD FORMAT IS 'SUBFILE1'
  287 029000         AT END SET NO-MORE-TRANSACTIONS-EXIST TO TRUE.
      029100 TEST-FOR-RECORD-INPUT-ERRORS.
  288 029200     READ SUBFILE PAYMENT-UPDATE-DISPLAY-FILE RECORD
      029300         FORMAT IS 'SUBFILE1'.
  289 029400     IF  STSCDE OF SUBFILE1-I IS EQUAL TO '1' THEN
  290 029500         SET INPUT-ERRORS-EXIST TO TRUE
      029600     ELSE
      029700         NEXT SENTENCE.
      029800 TRANSACTION-VALIDATION.
  291 029900     MOVE CUST OF SUBFILE1-I OF PAYMENT-UPDATE-DISPLAY-RECORD
```

Figure 37 (Part 11 of 14). Example Payment Update Program

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      030000         TO CUST OF CUSTOMER-MASTER-RECORD.
  292 030100     SET CUSTOMER-EXIST TO TRUE.
  293 030200     READ CUSTOMER-MASTER-FILE
  294 030300         INVALID KEY SET CUSTOMER-NOT-FOUND TO TRUE.
  295 030400     IF  CUSTOMER-EXIST THEN
  296 030500         MOVE CUST OF CUSMST TO CUST OF ORDHDR
  297 030600         MOVE ZEROES TO INVNUM
  298 030700         SET CUSTOMER-INVOICE-EXIST TO TRUE
  299 030800         PERFORM START-ON-CUSTOMER-INVOICE-FILE
  300 030900         IF  CUSTOMER-INVOICE-EXIST THEN
  301 031000             PERFORM READ-CUSTOMER-INVOICE-RECORD
  302 031100             IF CUSTOMER-INVOICE-EXIST THEN
  303 031200                 PERFORM CUSTOMER-MASTER-FILE-UPDATE
  304 031300                 MOVE AMPAID OF SUBFILE1-I TO AMOUNT-PAID
  305 031400                 SET PAYMENT-EXIST TO TRUE
  306 031500                 PERFORM PAYMENT-UPDATE
      031600                     UNTIL NO-MORE-INVOICES-EXIST
      031700                     OR NO-MORE-PAYMENT-EXIST
  307 031800                 IF  ARBAL OF CUSTOMER-MASTER-RECORD IS NEGATIVE
  308 031900                     SET MAKE-FIELD-BLINK
      032000                         DISPLAY-FIELD
      032100                         DO-NOT-REVERSE-FIELD-IMAGE
      032200                         OVER-PAYMENT-NOT-DISPLAYED
      032300                         DISPLAY-OVER-PAYMENT
      032400                         DISPLAY-EXCEPTION
      032500                         DO-NOT-DISPLAY-ACCEPT-PAYMENT
      032600                         PROTECT-INPUT-FIELD TO TRUE
  309 032700                     MOVE ARBAL TO OVRPMT OF SUBFILE1-O
  310 032800                     MOVE MESSAGE-THREE TO ECPMSG OF SUBFILE1-O
  311 032900                     MOVE '0' TO STSCDE OF SUBFILE1-O
  312 033000                     PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
      033100                 ELSE
  313 033200                     SET DO-NOT-DISPLAY-FIELD
      033300                         DO-NOT-DISPLAY-OVER-PAYMENT
      033400                         DO-NOT-REVERSE-FIELD-IMAGE
      033500                         DO-NOT-MAKE-FIELD-BLINK
      033600                         DO-NOT-DISPLAY-ACCEPT-PAYMENT
      033700                         PROTECT-INPUT-FIELD TO TRUE
  314 033800                     MOVE SPACES TO ECPMSG OF SUBFILE1-O
  315 033900                     MOVE ZEROES TO OVRPMT OF SUBFILE1-O
  316 034000                     MOVE '0' TO STSCDE OF SUBFILE1-O
  317 034100                     PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
      034200             ELSE
  318 034300                 PERFORM NO-CUSTOMER-INVOICE-ROUTINE
      034400         ELSE
  319 034500             PERFORM NO-CUSTOMER-INVOICE-ROUTINE
      034600     ELSE
  320 034700         SET REVERSE-FIELD-IMAGE
      034800             DO-NOT-PROTECT-INPUT-FIELD
      034900             DISPLAY-FIELD
      035000             DO-NOT-DISPLAY-OVER-PAYMENT
      035100             DO-NOT-MAKE-FIELD-BLINK
      035200             DISPLAY-EXCEPTION
      035300             DO-NOT-DISPLAY-ACCEPT-PAYMENT
      035400             DO-NOT-PROTECT-INPUT-FIELD TO TRUE
```

Figure 37 (Part 12 of 14). Example Payment Update Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  321  035500         MOVE ZEROES TO OVRPMT OF SUBFILE1-O
  322  035600         MOVE MESSAGE-ONE TO ECPMSG OF SUBFILE1-O
  323  035700         MOVE '1' TO STSCDE OF SUBFILE1-O
  324  035800         PERFORM REWRITE-DISPLAY-SUBFILE-RECORD.
  325  035900     PERFORM READ-MODIFIED-SUBFILE-RECORD.
       036000 START-ON-CUSTOMER-INVOICE-FILE.
  326  036100     START CUSTOMER-INVOICE-FILE
       036200         KEY IS GREATER THAN COMP-KEY
  327  036300         INVALID KEY SET NO-MORE-INVOICES-EXIST TO TRUE.
       036400 READ-CUSTOMER-INVOICE-RECORD.
  328  036500     READ CUSTOMER-INVOICE-FILE NEXT RECORD
  329  036600         AT END SET NO-MORE-INVOICES-EXIST TO TRUE.
  330  036700     IF  CUST OF CUSTOMER-MASTER-RECORD
       036800         IS NOT EQUAL TO CUST OF CUSTOMER-INVOICE-RECORD THEN
  331  036900         SET NO-MORE-INVOICES-EXIST TO TRUE
       037000     ELSE
       037100         NEXT SENTENCE.
       037200 CUSTOMER-MASTER-FILE-UPDATE.
  332  037300     MOVE FILE-DATE TO LSTDAT OF CUSTOMER-MASTER-RECORD.
  333  037400     MOVE AMPAID OF SUBFILE1-I
       037500         TO LSTAMT OF CUSTOMER-MASTER-RECORD.
  334  037600     SUBTRACT AMPAID OF SUBFILE1-I
       037700         FROM ARBAL OF CUSTOMER-MASTER-RECORD.
  335  037800     REWRITE CUSTOMER-MASTER-RECORD.
       037900  REWRITE-DISPLAY-SUBFILE-RECORD.
  336  038000     MOVE AMPAID OF SUBFILE1-I TO AMPAID OF SUBFILE1-O.
  337  038100     MOVE CUST OF SUBFILE1-I TO CUST OF SUBFILE1-O.
  338  038200     SET WRITE-DISPLAY TO TRUE.
  339  038300     SET SUBFILE1-FORMAT TO TRUE.
  340  038400     MOVE CORR INDICATOR-AREA TO SUBFILE1-O-INDIC.
  341  038500     REWRITE SUBFILE PAYMENT-UPDATE-DISPLAY-RECORD
       038600         FORMAT IS 'SUBFILE1'.
       038700 NO-CUSTOMER-INVOICE-ROUTINE.
  342  038800     IF  STSCDE OF SUBFILE1-I IS EQUAL TO '1' THEN
  343  038900        IF  ACPPMT OF SUBFILE1-I IS EQUAL TO '*NO' THEN
  344  039000           SET DO-NOT-DISPLAY-FIELD
       039100               DO-NOT-DISPLAY-OVER-PAYMENT
       039200               DO-NOT-REVERSE-FIELD-IMAGE
       039300               DO-NOT-MAKE-FIELD-BLINK
       039400               DO-NOT-DISPLAY-ACCEPT-PAYMENT
       039500               PROTECT-INPUT-FIELD
       039600                 TO TRUE
  345  039700            MOVE SPACES TO ECPMSG OF SUBFILE1-O
  346  039800            MOVE ZEROES TO OVRPMT OF SUBFILE1-O
  347  039900            MOVE '0' TO STSCDE OF SUBFILE1-O
  348  040000            PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
       040100        ELSE
  349  040200            PERFORM CUSTOMER-MASTER-FILE-UPDATE
  350  040300            SET MAKE-FIELD-BLINK
       040400                DISPLAY-FIELD
       040500                DO-NOT-REVERSE-FIELD-IMAGE
       040600                OVER-PAYMENT-NOT-DISPLAYED
       040700                DISPLAY-OVER-PAYMENT
       040800                DISPLAY-EXCEPTION
       040900                DO-NOT-DISPLAY-ACCEPT-PAYMENT
```

Figure 37 (Part 13 of 14). Example Payment Update Program

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      041000                PROTECT-INPUT-FIELD
      041100                  TO TRUE
  351 041200            MOVE ARBAL TO OVRPMT OF SUBFILE1-O
  352 041300            MOVE MESSAGE-THREE TO ECPMSG OF SUBFILE1-O
  353 041400            MOVE '0' TO STSCDE OF SUBFILE1-O
  354 041500            PERFORM REWRITE-DISPLAY-SUBFILE-RECORD
      041600     ELSE
  355 041700         SET REVERSE-FIELD-IMAGE
      041800             DISPLAY-FIELD
      041900             DO-NOT-PROTECT-INPUT-FIELD
      042000             DO-NOT-DISPLAY-OVER-PAYMENT
      042100             DISPLAY-EXCEPTION
      042200             DISPLAY-ACCEPT-PAYMENT
      042300             DISPLAY-ACCEPT-HEADING
      042400             DISPLAY-ACCEPT-MESSAGE
      042500             DO-NOT-MAKE-FIELD-BLINK
      042600               TO TRUE
  356 042700         MOVE ZEROS TO OVRPMT OF SUBFILE1-O
  357 042800         MOVE MESSAGE-TWO TO ECPMSG OF SUBFILE1-O
  358 042900         MOVE '1' TO STSCDE OF SUBFILE1-O
  359 043000         PERFORM REWRITE-DISPLAY-SUBFILE-RECORD.
      043100 PAYMENT-UPDATE.
  360 043200     SUBTRACT AMPAID OF CUSTOMER-INVOICE-RECORD
      043300         FROM ORDAMT OF CUSTOMER-INVOICE-RECORD
      043400         GIVING AMOUNT-OWED.
  361 043500     SUBTRACT AMOUNT-PAID
      043600         FROM AMOUNT-OWED
      043700         GIVING INVOICE-BALANCE.
  362 043800     IF  INVOICE-BALANCE IS LESS THAN .01 THEN
  363 043900         MOVE 2 TO OPNSTS OF CUSTOMER-INVOICE-RECORD
  364 044000         MOVE ORDAMT OF CUSTOMER-INVOICE-RECORD
      044100           TO AMPAID OF CUSTOMER-INVOICE-RECORD
  365 044200         SUBTRACT AMOUNT-OWED
      044300             FROM AMOUNT-PAID
      044400     ELSE
  366 044500         ADD AMOUNT-PAID TO AMPAID OF CUSTOMER-INVOICE-RECORD
  367 044600         SET NO-MORE-PAYMENT-EXIST TO TRUE.
  368 044700     REWRITE CUSTOMER-INVOICE-RECORD.
  369 044800     IF  NO-MORE-PAYMENT-EXIST THEN
      044900         NEXT SENTENCE
      045000     ELSE
  370 045100         PERFORM READ-CUSTOMER-INVOICE-RECORD.
      045200 INITIALIZE-SUBFILE-RECORD.
  371 045300     MOVE SPACES TO CUST OF SUBFILE1-O.
  372 045400     MOVE SPACES TO ECPMSG OF SUBFILE1-O.
  373 045500     MOVE '0' TO STSCDE OF SUBFILE1-O.
  374 045600     MOVE ZEROS TO AMPAID OF SUBFILE1-O.
  375 045700     MOVE ZEROS TO OVRPMT OF SUBFILE1-O.
  376 045800     ADD 1 TO REL-NUMBER.
  377 045900     MOVE CORR INDICATOR-AREA TO SUBFILE1-O-INDIC.
  378 046000     WRITE SUBFILE PAYMENT-UPDATE-DISPLAY-RECORD
      046100         FORMAT IS 'SUBFILE1'.
      046200 CLEAN-UP-ROUTINE.
  379 046300     CLOSE CUSTOMER-INVOICE-FILE
      046400           CUSTOMER-MASTER-FILE
      046500           PAYMENT-UPDATE-DISPLAY-FILE.
  380 046600     STOP RUN.
                        * * * * *  E N D  O F  S O U R C E  * * * * *
```

Figure 37 (Part 14 of 14). Example Payment Update Program

This is the initial display that is written to the work station to prompt the user to enter the customer number and payment:

```
Customer Payment Update Prompt                                  Date  03/02/94

             Customer    Payment

               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
               _____   __________
```

The user enters the customer numbers and payments:

```
Customer Payment Update Prompt                                  Date  03/02/94

             Customer    Payment

              34500      2000
              40500     30000
              36000      2500
              12500       200
              22799      4500
              41900      7500
              10001      5000
              49500      2500
              13300      3500
              56900      4000
```

Payments that would result in overpayments or that have invalid customer numbers are left on the display and appropriate messages are added:

```
  Customer Payment Update Prompt                                  Date  03/02/94

  Accept      Customer    Payment     Exception message
  Payment

    ____       40500       30000      NO INVOICES EXIST FOR CUSTOMER

    ____       12500         200      NO INVOICES EXIST FOR CUSTOMER

    ____       41900        7500      NO INVOICES EXIST FOR CUSTOMER
               10001        5000      CUSTOMER DOES NOT EXIST

    ____       13300        3500      NO INVOICES EXIST FOR CUSTOMER


Accept payment values: (*NO *YES)
```

The user indicates which payments to accept:

```
  Customer Payment Update Prompt                                  Date  03/02/94

  Accept      Customer    Payment     Exception message
  Payment

    *NO        40500       30000      NO INVOICES EXIST FOR CUSTOMER

    *YES       12500         200      NO INVOICES EXIST FOR CUSTOMER

    *NO        41900        7500      NO INVOICES EXIST FOR CUSTOMER
               10001        5000      CUSTOMER DOES NOT EXIST

    *NO        13300        3500      NO INVOICES EXIST FOR CUSTOMER


Accept payment values: (*NO *YES)
```

The accepted payments are processed and overpayment information is displayed:

```
Customer Payment Update Prompt                                  Date  03/02/94

Accept      Customer    Payment     Exception message
Payment


             12500         200      NO INVOICES EXIST FOR CUSTOMER

             10001        5000      CUSTOMER DOES NOT EXIST
```

End of IBM Extension

Chapter 6. Example Programs

The programs in this chapter illustrate the fundamental programming techniques associated with each type of file organization. They are intended to be used for planning purposes only, and to illustrate the input/output statements necessary for certain access methods. Other COBOL features are used only incidentally. The programs are:

- Sequential File Creation
- Sequential File Updating and Extension
- Indexed File Creation
- Indexed File Updating
- Relative File Creation
- Relative File Updating
- Relative File Retrieval.

Sequential File Creation

This program creates a sequential file of employee salary records. The input records are arranged in ascending order of employee number. The output file has the identical order.

```
5763CB1                              COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.  CREATESEQ.
    3  000300
    4  000400 ENVIRONMENT DIVISION.
    5  000500 CONFIGURATION SECTION.
    6  000600 SOURCE-COMPUTER.   IBM-S38.
    7  000700 OBJECT-COMPUTER.   IBM-S38.
    8  000800 SPECIAL-NAMES.  CONSOLE IS TYPEWRITER.
    9  000900 INPUT-OUTPUT SECTION.
   10  001000 FILE-CONTROL.
   11  001100     SELECT INPUT-FILE ASSIGN TO DISK-FILEA
   12  001200         FILE STATUS IS INPUT-FILE-STATUS.
   13  001300     SELECT OUTPUT-FILE ASSIGN TO DISK-FILEB
   14  001400         FILE STATUS IS OUTPUT-FILE-STATUS.
   15  001500 DATA DIVISION.
   16  001600 FILE SECTION.
   17  001700 FD  INPUT-FILE LABEL RECORDS STANDARD.
   18  001800 01  INPUT-RECORD.
   19  001900     05  INPUT-EMPLOYEE-NUMBER      PICTURE 9(6).
   20  002000     05  INPUT-EMPLOYEE-NAME        PICTURE X(28).
   21  002100     05  INPUT-EMPLOYEE-CODE        PICTURE 9.
   22  002200     05  INPUT-EMPLOYEE-SALARY      PICTURE 9(6)V99.
   23  002300 FD  OUTPUT-FILE LABEL RECORDS STANDARD.
   24  002400 01  OUTPUT-RECORD.
   25  002500     05  OUTPUT-EMPLOYEE-NUMBER     PICTURE 9(6).
   26  002600     05  OUTPUT-EMPLOYEE-NAME       PICTURE X(28).
   27  002700     05  OUTPUT-EMPLOYEE-CODE       PICTURE 9.
   28  002800     05  OUTPUT-EMPLOYEE-SALARY     PICTURE 9(6)V99.
```

Figure 38 (Part 1 of 3). Example of Sequential File Creation

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   29  002900 WORKING-STORAGE SECTION.
   30  003000 77  INPUT-FILE-STATUS              PICTURE XX.
   31  003100 77  OUTPUT-FILE-STATUS             PICTURE XX.
   32  003200 01  INPUTEND                       PICTURE X VALUE SPACE.
   33  003300     88  THE-END-OF-INPUT           VALUE "E".
   34  003400 01  DISP-RECORD.
   35  003500     05  OP-NAME                    PICTURE X(7).
   36  003600     05  FILLER                     PICTURE XX VALUE SPACE.
   37  003700     05  FILE-NAME                  PICTURE X(11).
   38  003800     05  FILLER                     PICTURE XX VALUE SPACE.
   39  003900     05  FILLER                     PICTURE X(14)
   40  004000                                    VALUE "FILE STATUS IS".
   41  004100     05  FILLER                     PICTURE XX VALUE SPACE.
   42  004200     05  SK                         PICTURE XX.
   43  004300 PROCEDURE DIVISION.
       004400 DECLARATIVES.
       004500 I-O-ERROR SECTION.
       004600           USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE,
       004700                                               OUTPUT-FILE.
       004800 I-O-ERROR-PARA.
       004900*************************************************************
       005000* DUMMY DECLARATIVES TO ENSURE CONTROL IS RETURNED TO THIS *
       005100* PROGRAM WHEN AN ERROR OCCURS DURING FILE PROCESSING.     *
       005200* ERROR HANDLING IS DONE AFTER EACH I/O STATEMENT.         *
       005300*************************************************************
       005400 END DECLARATIVES.
       005500 MAIN-PROGRAM SECTION.
       005600 OPEN-FILES.
   44  005700     OPEN INPUT INPUT-FILE
       005800         OUTPUT OUTPUT-FILE.
   45  005900     IF INPUT-FILE-STATUS NOT = "00"
   46  006000         MOVE "OPEN" TO OP-NAME
   47  006100         MOVE "INPUT-FILE" TO FILE-NAME
   48  006200         MOVE INPUT-FILE-STATUS TO SK
   49  006300         PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2.
   50  006400     IF OUTPUT-FILE-STATUS NOT = "00"
   51  006500        MOVE "OPEN" TO OP-NAME
   52  006600        MOVE "OUTPUT-FILE" TO FILE-NAME
   53  006700        MOVE OUTPUT-FILE-STATUS TO SK
   54  006800        PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2.
   55  006900     PERFORM BUILD-FILE UNTIL THE-END-OF-INPUT.
       007000 CLOSE-FILES.
   56  007100        CLOSE INPUT-FILE
       007200              OUTPUT-FILE.
   57  007300        STOP RUN.
       007400 BUILD-FILE.
   58  007500     READ INPUT-FILE INTO OUTPUT-RECORD
   59  007600         AT END SET THE-END-OF-INPUT TO TRUE.
   60  007700     IF INPUT-FILE-STATUS NOT = "00"
   61  007800           MOVE "WRITE" TO OP-NAME
   62  007900           MOVE "OUTPUT-FILE" TO FILE-NAME
   63  008000           MOVE OUTPUT-FILE-STATUS TO SK
   64  008100           PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2
   65  008200           GO TO CLOSE-FILES.
```

Figure 38 (Part 2 of 3). Example of Sequential File Creation

```
5763CB1                              COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   66  008300       WRITE OUTPUT-RECORD.
   67  008400       IF OUTPUT-FILE-STATUS NOT = "00"
   68  008500           MOVE "WRITE" TO OP-NAME
   69  008600           MOVE "OUTPUT-FILE" TO FILE-NAME
   70  008700           MOVE OUTPUT-FILE-STATUS TO SK
   71  008800           PERFORM ERROR-OUT-1 THROUGH ERROR-OUT-2
   72  008900           GO TO CLOSE-FILES.
       009000 ERROR-OUT-1.
   73  009100           DISPLAY "FILE PROCESSING ERROR" UPON TYPEWRITER.
   74  009200           DISPLAY DISP-RECORD UPON TYPEWRITER.
   75  009300           CLOSE INPUT-FILE
       009400                 OUTPUT-FILE.
   76  009500           STOP RUN.
       009600 ERROR-OUT-2.
       009700     EXIT.
                        * * * * *  E N D  O F  S O U R C E  * * * * *
5763CB1                               COBOL MESSAGES
 STMT
*   17  MSGID: CBL0650  SEVERITY: 00  SEQNBR:  001700
       Message . . . . :   Blocking/Deblocking for file 'INPUT-FILE'
         will be performed by compiler-generated code.
*   23  MSGID: CBL0650  SEVERITY: 00  SEQNBR:  002300
       Message . . . . :   Blocking/Deblocking for file 'OUTPUT-FILE'
         will be performed by compiler-generated code.
*   44  MSGID: CBL0335  SEVERITY: 00  SEQNBR:  005400
       Message . . . . :   Empty paragraph or section precedes 'END
         DECLARATIVES' paragraph or section.
                                      MESSAGE SUMMARY
 TOTAL    INFO(0-4)    WARNING(5-19)    ERROR(20-29)    SEVERE(30-39)    TERMINAL(40-99)
     3          3            0               0               0                0
                 * * * * *  E N D  O F  C O B O L  M E S S A G E S  * * * * *
```

Figure 38 (Part 3 of 3). Example of Sequential File Creation

Sequential File Updating and Extension

This program updates and extends the file created by the CREATESEQ program. The INPUT-FILE and the MASTER-FILE are each read. When a match is found between INPUT-EMPLOYEE-NUMBER and MST-EMPLOYEE-NUMBER, the input record replaces the original record. After the MASTER-FILE has been completely processed, new employee records are added at the end of the file.

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.  UPDATESEQ.
    3  000300 ENVIRONMENT DIVISION.
    4  000400 CONFIGURATION SECTION.
    5  000500 SOURCE-COMPUTER.  IBM-S38.
    6  000600 OBJECT-COMPUTER.  IBM-S38.
    7  000700 INPUT-OUTPUT SECTION.
    8  000800 FILE-CONTROL.
    9  000900     SELECT INPUT-FILE ASSIGN TO DISK-FILEB
   10  001000         FILE STATUS IS INPUT-FILE-STATUS.
   11  001100     SELECT MASTER-FILE ASSIGN TO DISK-MSTFILEB
   12  001200         FILE STATUS IS MASTER-FILE-STATUS.
   13  001300
   14  001400 DATA DIVISION.
   15  001500 FILE SECTION.
   16  001600 FD  INPUT-FILE LABEL RECORDS STANDARD.
   17  001700 01  INPUT-RECORD.
   18  001800     05  INPUT-EMPLOYEE-NUMBER      PICTURE 9(6).
   19  001900     05  INPUT-EMPLOYEE-NAME        PICTURE X(28).
   20  002000     05  INPUT-EMPLOYEE-CODE        PICTURE 9.
   21  002100     05  INPUT-EMPLOYEE-SALARY      PICTURE 9(6)V99.
   22  002200 FD  MASTER-FILE LABEL RECORDS STANDARD.
   23  002300 01  MASTER-RECORD.
   24  002400     05 MST-EMPLOYEE-NUMBER         PICTURE 9(6).
   25  002500     05 MST-EMPLOYEE-NAME           PICTURE X(28).
   26  002600     05 MST-EMPLOYEE-CODE           PICTURE 9.
   27  002700     05 MST-EMPLOYEE-SALARY         PICTURE 9(6)V99.
   28  002800 WORKING-STORAGE SECTION.
   29  002900 77  INPUT-FILE-STATUS             PICTURE XX.
   30  003000 77  MASTER-FILE-STATUS            PICTURE XX.
   31  003100 01  INPUTEND                      PICTURE X VALUE SPACE.
   32  003200     88 THE-END-OF-INPUT           VALUE "E".
   33  003300 01  MASTEREND                     PICTURE X VALUE SPACE.
   34  003400     88 THE-END-OF-MASTER          VALUE "E".
   35  003500 01  ERROR-INFO.
   36  003600     05 OP-NAME                     PICTURE X(12).
   37  003700     05 FILLER                      PICTURE XX VALUE SPACE.
   38  003800     05 FILE-NAME                   PICTURE X(11).
   39  003900     05 FILLER                      PICTURE XX VALUE SPACE.
   40  004000     05 FILLER                      PICTURE X(14)
   41  004100                                   VALUE "FILE STATUS IS".
   42  004200     05 FILLER                      PICTURE XX VALUE SPACE.
   43  004300     05 SK                          PICTURE XX.
   44  004400 PROCEDURE DIVISION.
       004500 DECLARATIVES.
       004600 INPUT-FILE-ERROR SECTION.
       004700      USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.
       004800 INPUT-FILE-ERROR-PARA.
   45  004900        MOVE INPUT-FILE-STATUS TO SK.
   46  005000        MOVE "INPUT-FILE" TO FILE-NAME.
   47  005100        DISPLAY "FILE PROCESSING ERROR".
   48  005200        DISPLAY ERROR-INFO.
   49  005300        DISPLAY "PROCESSING TERMINATED DUE TO I-O ERROR".
   50  005400        STOP RUN.
```

Figure 39 (Part 1 of 2). Example of Sequential File Updating and Extension

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      005500 I-O-FILE-ERROR SECTION.
      005600     USE AFTER STANDARD ERROR PROCEDURE ON MASTER-FILE.
      005700 MASTER-FILE-ERROR-PARA.
   51 005800        MOVE MASTER-FILE-STATUS TO SK.
   52 005900        MOVE "MASTER-FILE" TO FILE-NAME.
   53 006000        DISPLAY "FILE PROCESSING ERROR".
   54 006100        DISPLAY ERROR-INFO.
   55 006200        DISPLAY "PROCESSING TERMINATED DUE TO I-O ERROR".
   56 006300        STOP RUN.
      006400 END DECLARATIVES.
      006500 MAIN-PROGRAM SECTION.
      006600 OPEN-FILES.
   57 006700        MOVE "OPEN" TO OP-NAME.
   58 006800        OPEN INPUT INPUT-FILE
      006900             I-O   MASTER-FILE.
      007000 PROCESSING-LOGIC.
   59 007100       PERFORM READ-INPUT-FILE.
   60 007200       PERFORM READ-MASTER-FILE.
   61 007300       PERFORM PROCESS-FILES UNTIL THE-END-OF-INPUT.
      007400 CLOSE-FILES.
   62 007500       MOVE "CLOSE" TO OP-NAME.
   63 007600       CLOSE MASTER-FILE
      007700             INPUT-FILE.
   64 007800       STOP RUN.
      007900 READ-INPUT-FILE.
   65 008000       MOVE "READ" TO OP-NAME.
   66 008100       READ INPUT-FILE
   67 008200           AT END SET THE-END-OF-INPUT TO TRUE.
      008300 READ-MASTER-FILE.
   68 008400       MOVE "READ" TO OP-NAME.
   69 008500       READ MASTER-FILE
      008600           AT END
   70 008700              SET THE-END-OF-MASTER TO TRUE
   71 008800              MOVE "AT END CLOSE" TO OP-NAME
   72 008900              CLOSE MASTER-FILE
   73 009000              MOVE "OPEN EXTEND" TO OP-NAME
   74 009100              OPEN EXTEND MASTER-FILE.
      009200 PROCESS-FILES.
   75 009300     IF THE-END-OF-MASTER
   76 009400         WRITE MASTER-RECORD FROM INPUT-RECORD
   77 009500         PERFORM READ-INPUT-FILE
      009600     ELSE
   78 009700         IF MST-EMPLOYEE-NUMBER LESS THAN INPUT-EMPLOYEE-NUMBER
   79 009800            PERFORM READ-MASTER-FILE
      009900        ELSE
   80 010000           IF MST-EMPLOYEE-NUMBER = INPUT-EMPLOYEE-NUMBER
   81 010100              MOVE "REWRITE" TO OP-NAME
   82 010200              REWRITE MASTER-RECORD FROM INPUT-RECORD
   83 010300              PERFORM READ-INPUT-FILE
   84 010400              PERFORM READ-MASTER-FILE
      010500           ELSE
   85 010600               DISPLAY "ERROR RECORD -> ", INPUT-EMPLOYEE-NUMBER
   86 010700               PERFORM READ-INPUT-FILE.
                       * * * * *  E N D  O F  S O U R C E  * * * * *
5763CB1                              COBOL MESSAGES
 STMT
*   16  MSGID: CBL0650  SEVERITY: 00  SEQNBR:  001600
       Message . . . . :   Blocking/Deblocking for file 'INPUT-FILE'
         will be performed by compiler-generated code.
                                    MESSAGE SUMMARY
 TOTAL    INFO(0-4)    WARNING(5-19)    ERROR(20-29)    SEVERE(30-39)    TERMINAL(40-99)
     1          1            0               0               0                0
              * * * * *  E N D  O F  C O B O L  M E S S A G E S  * * * * *
```

Figure 39 (Part 2 of 2). Example of Sequential File Updating and Extension

Indexed File Creation

This program creates an indexed file of summary records for bank depositors. The key within each input indexed file record is `INPUT-KEY` (the depositor's account number); the input records are ordered in ascending sequence upon this key. Records are read from the input file and transferred to the indexed file record area. The indexed file record is then written.

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.  CREATEIND.
    3  000300
    4  000400 ENVIRONMENT DIVISION.
    5  000500 CONFIGURATION SECTION.
    6  000600 SOURCE-COMPUTER. IBM-S38.
    7  000700 OBJECT-COMPUTER. IBM-S38.
    8  000800 INPUT-OUTPUT SECTION.
    9  000900 FILE-CONTROL.
   10  001000     SELECT INDEXED-FILE ASSIGN TO DISK-INDEXFILE
   11  001100         ORGANIZATION IS INDEXED
   12  001200         ACCESS IS SEQUENTIAL
   13  001300         RECORD KEY IS INDEX-KEY
   14  001400         FILE STATUS IS INDEXED-FILE-STATUS.
   15  001500     SELECT INPUT-FILE ASSIGN TO DISK-FILEG
   16  001600         FILE STATUS IS INPUT-FILE-STATUS.
   17  001700 DATA DIVISION.
   18  001800 FILE SECTION.
   19  001900 FD  INDEXED-FILE LABEL RECORDS STANDARD.
   20  002000 01  INDEX-RECORD.
   21  002100     05  INDEX-KEY                    PICTURE X(10).
   22  002200     05  INDEX-FLD1                   PICTURE X(10).
   23  002300     05  INDEX-NAME                   PICTURE X(20).
   24  002400     05  INDEX-BAL                    PICTURE S9(5)V99.
   25  002500 FD  INPUT-FILE LABEL RECORDS STANDARD.
   26  002600 01  INPUT-RECORD.
   27  002700     05  INPUT-KEY                    PICTURE X(10).
   28  002800     05  INPUT-NAME                   PICTURE X(20).
   29  002900     05  INPUT-BAL                    PICTURE S9(5)V99.
   30  003000 WORKING-STORAGE SECTION.
   31  003100 77  INDEXED-FILE-STATUS              PICTURE XX.
   32  003200 77  INPUT-FILE-STATUS                PICTURE XX.
   33  003300 77  OP-NAME                          PICTURE X(7).
   34  003400 01  INPUTEND                         PICTURE X VALUE SPACES.
   35  003500     88  THE-END-OF-INPUT             VALUE "E".
   36  003600 01  ERRORFLAG                        PICTURE X VALUE SPACES.
   37  003700     88  ERROR-OCCURRED               VALUE "1".
   38  003800 PROCEDURE DIVISION.
       003900 DECLARATIVES.
       004000 INPUT-ERROR SECTION.
       004100       USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
       004200 INPUT-ERROR-PARA.
   39  004300     DISPLAY "UNEXPECTED ERROR ON ", OP-NAME, " FOR INPUT-FILE ".
   40  004400     DISPLAY "FILE STATUS IS ", INPUT-FILE-STATUS.
   41  004500     SET ERROR-OCCURRED TO TRUE.
       004600 OUTPUT-ERROR SECTION.
       004700       USE AFTER STANDARD ERROR PROCEDURE ON OUTPUT.
       004800 OUTPUT-ERROR-PARA.
   42  004900     DISPLAY "UNEXPECTED ERROR ON ", OP-NAME, " FOR INDEXED-FILE ".
   43  005000     DISPLAY "FILE STATUS IS ", INDEXED-FILE-STATUS.
   44  005100     SET ERROR-OCCURRED TO TRUE.
       005200 END DECLARATIVES.
       005300 MAIN-PROCESSING SECTION.
       005400 MAIN-PROCEDURE.
   45  005500     MOVE "OPEN" TO OP-NAME.
```

Figure 40 (Part 1 of 2). Example of Indexed File Creation

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   46  005600     OPEN INPUT INPUT-FILE
       005700         OUTPUT INDEXED-FILE.
   47  005800     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
   49  005900     PERFORM READ-INPUT-FILE.
   50  006000     PERFORM LOAD-INDEXED-FILE THRU READ-INPUT-FILE
       006100                               UNTIL THE-END-OF-INPUT.
   51  006200     MOVE "CLOSE" TO OP-NAME.
   52  006300     CLOSE INPUT-FILE
       006400           INDEXED-FILE.
   53  006500     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
   55  006600     STOP RUN.
       006700 LOAD-INDEXED-FILE.
   56  006800     MOVE INPUT-KEY TO INDEX-KEY.
   57  006900     MOVE INPUT-NAME TO INDEX-NAME.
   58  007000     MOVE INPUT-BAL TO INDEX-BAL.
   59  007100     MOVE SPACES TO INDEX-FLD1.
   60  007200     MOVE "WRITE" TO OP-NAME.
   61  007300     WRITE INDEX-RECORD
       007400           INVALID KEY
   62  007500               DISPLAY "WRITE FAILED FOR KEY ", INDEX-KEY.
   63  007600     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
       007700 READ-INPUT-FILE.
   65  007800     MOVE "READ" TO OP-NAME.
   66  007900     READ INPUT-FILE
   67  008000          AT END SET THE-END-OF-INPUT TO TRUE.
   68  008100     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
       008200 ERROR-TERMINATION.
   70  008300     DISPLAY "I-O ERROR OCCURRED - PROCESS TERMINATING".
   71  008400     STOP RUN.
                         * * * * *  E N D  O F  S O U R C E  * * * * *
 5763CB1                              COBOL MESSAGES
  STMT
*   19  MSGID: CBL0650  SEVERITY: 00  SEQNBR:  001900
       Message . . . . :   Blocking/Deblocking for file 'INDEXED-FILE'
         will be performed by compiler-generated code.
*   25  MSGID: CBL0650  SEVERITY: 00  SEQNBR:  002500
       Message . . . . :   Blocking/Deblocking for file 'INPUT-FILE'
         will be performed by compiler-generated code.
                                     MESSAGE SUMMARY
  TOTAL    INFO(0-4)    WARNING(5-19)    ERROR(20-29)    SEVERE(30-39)    TERMINAL(40-99)
     2          2            0               0               0                0
                  * * * * *  E N D  O F  C O B O L  M E S S A G E S  * * * * *
```

Figure 40 (Part 2 of 2). Example of Indexed File Creation

Indexed File Updating

This program, using dynamic access, updates the indexed file created in the CREATEIND program.

The input records contain the key for the record, the depositor name, and the amount of the transaction.

When the input record is read, the program tests whether this is a transaction record (in which case, all fields of the record are filled) or a record requesting sequential retrieval of a specific generic class (in which case, only the INPUT-GEN-FLD of the input record contains data).

Random access is used for the updating and printing of the transaction records. Sequential access is used for the retrieval and printing of all records within one generic class.

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.  UPDATEIND.
    3  000300
    4  000400 ENVIRONMENT DIVISION.
    5  000500 CONFIGURATION SECTION.
    6  000600 SOURCE-COMPUTER. IBM-S38.
    7  000700 OBJECT-COMPUTER. IBM-S38.
    8  000800 INPUT-OUTPUT SECTION.
    9  000900 FILE-CONTROL.
   10  001000     SELECT MASTER-FILE ASSIGN TO DISK-INDEXFILE
   11  001100         ORGANIZATION IS INDEXED
   12  001200         ACCESS IS DYNAMIC
   13  001300         RECORD KEY IS MASTER-KEY
   14  001400         FILE STATUS IS MASTER-FILE-STATUS.
   15  001500     SELECT INPUT-FILE ASSIGN TO DISK-FILEH
   16  001600         FILE STATUS IS INPUT-FILE-STATUS.
   17  001700     SELECT PRINT-FILE ASSIGN TO PRINTER-OSYSPRT
   18  001800         FILE STATUS IS PRINT-FILE-STATUS.
   19  001900 DATA DIVISION.
   20  002000 FILE SECTION.
   21  002100 FD  MASTER-FILE LABEL RECORDS STANDARD.
   22  002200 01  MASTER-RECORD.
   23  002300     05  MASTER-KEY.
   24  002400       10    MASTER-GEN-FLD   PICTURE X(5).
   25  002500       10    MASTER-DET-FLD   PICTURE X(5).
   26  002600     05  MASTER-FLD1          PICTURE X(10).
   27  002700     05  MASTER-NAME          PICTURE X(20).
   28  002800     05  MASTER-BAL           PICTURE S9(5)V99.
   29  002900 FD  INPUT-FILE LABEL RECORDS STANDARD.
   30  003000 01  INPUT-REC.
   31  003100     05  INPUT-KEY.
   32  003200       10  INPUT-GEN-FLD      PICTURE X(5).
   33  003300       10  INPUT-DET-FLD      PICTURE X(5).
   34  003400     05  INPUT-NAME           PICTURE X(20).
   35  003500     05  INPUT-AMT            PICTURE S9(5)V99.
   36  003600 FD  PRINT-FILE LABEL RECORDS OMITTED
   37  003700     LINAGE 12 LINES FOOTING AT 9.
```

Figure 41 (Part 1 of 5). Example of Indexed File Updating

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   38  003800 01  PRINT-RECORD-1.
   39  003900     05  PRINT-KEY            PICTURE X(10).
   40  004000     05  FILLER               PICTURE X(5).
   41  004100     05  PRINT-NAME           PICTURE X(20).
   42  004200     05  FILLER               PICTURE X(5).
   43  004300     05  PRINT-BAL            PICTURE $$$,$$9.99-.
   44  004400     05  FILLER               PICTURE X(7).
   45  004500     05  PRINT-AMT            PICTURE $$$,$$9.99-.
   46  004600     05  FILLER               PICTURE X(5).
   47  004700     05  PRINT-NEW-BAL        PICTURE $$$,$$9.99-.
   48  004800 01  PRINT-RECORD-2           PICTURE X(89).
   49  004900 WORKING-STORAGE SECTION.
   50  005000 77  MASTER-FILE-STATUS       PICTURE XX.
   51  005100 77  INPUT-FILE-STATUS        PICTURE XX.
   52  005200 77  PRINT-FILE-STATUS        PICTURE XX.
   53  005300 77  LINES-TO-FOOT            PICTURE 99.
   54  005400 01  PAGE-HEAD.
   55  005500     05  FILLER               PICTURE X(38) VALUE SPACES.
   56  005600     05  FILLER               PICTURE X(13) VALUE "UPDATE REPORT".
   57  005700     05  FILLER               PICTURE X(38) VALUE SPACES.
   58  005800 01  COLUMN-HEAD.
   59  005900     05  FILLER               PICTURE X(6)  VALUE "KEY ID".
   60  006000     05  FILLER               PICTURE X(9)  VALUE SPACES.
   61  006100     05  FILLER               PICTURE X(4)  VALUE "NAME".
   62  006200     05  FILLER               PICTURE X(21) VALUE SPACES.
   63  006300     05  FILLER               PICTURE X(11) VALUE "CUR BALANCE".
   64  006400     05  FILLER               PICTURE X(6)  VALUE SPACES.
   65  006500     05  FILLER               PICTURE X(13) VALUE "UPDATE AMOUNT".
   66  006600     05  FILLER               PICTURE X(4)  VALUE SPACES.
   67  006700     05  FILLER               PICTURE X(11) VALUE "NEW BALANCE".
   68  006800     05  FILLER               PICTURE X(4)  VALUE SPACES.
   69  006900 01  PAGE-FOOT.
   70  007000     05  FILLER               PICTURE X(81)  VALUE SPACES.
   71  007100     05  FILLER               PICTURE A(6)   VALUE "PAGE  ".
   72  007200     05  PG-NUMBER            PICTURE 99     VALUE 00.
   73  007300
   74  007400 01  INPUTEND                 PICTURE X VALUE SPACE.
   75  007500     88  THE-END-OF-INPUT               VALUE "E".
   76  007600 01  ERRORFLAG                PICTURE X VALUE SPACE.
   77  007700     88  ERROR-OCCURRED                 VALUE "1".
   78  007800 01  ERROR-DATA.
   79  007900     05  FILLER               PICTURE X(21)
   80  008000                               VALUE "STATEMENT FAILING IS ".
   81  008100     05  OP-NAME              PICTURE X(9).
   82  008200     05  FILLER               PICTURE X(16)
   83  008300                               VALUE "FILE STATUS IS".
   84  008400     05  STATUS-VALUE         PICTURE XX.
   85  008500 01  INPUT-MESSAGE.
   86  008600     05  FILLER               PICTURE X(30)
   87  008700         VALUE "UNEXPECTED ERROR ON INPUT-FILE" .
   88  008800 01  I-O-MESSAGE.
   89  008900     05  FILLER               PICTURE X(31)
   90  009000         VALUE "UNEXPECTED ERROR ON MASTER-FILE" .
   91  009100 01  OUTPUT-MESSAGE.
   92  009200     05  FILLER               PICTURE X(30)
   93  009300         VALUE "UNEXPECTED ERROR ON PRINT-FILE" .
```

Figure 41 (Part 2 of 5). Example of Indexed File Updating

```
5763CB1                              COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   94  009400 PROCEDURE DIVISION.
       009500 DECLARATIVES.
       009600 INPUT-ERROR SECTION.
       009700       USE AFTER STANDARD ERROR PROCEDURE ON INPUT.
       009800 INPUT-ERROR-PARA.
   95  009900       DISPLAY INPUT-MESSAGE.
   96  010000       MOVE INPUT-FILE-STATUS TO STATUS-VALUE.
   97  010100       DISPLAY ERROR-DATA.
   98  010200       SET ERROR-OCCURRED TO TRUE.
       010300 I-O-ERROR SECTION.
       010400     USE AFTER STANDARD ERROR PROCEDURE ON I-O.
       010500 I-O-ERROR-PARA.
   99  010600       DISPLAY I-O-MESSAGE.
  100  010700       MOVE MASTER-FILE-STATUS TO STATUS-VALUE.
  101  010800       DISPLAY ERROR-DATA.
  102  010900       SET ERROR-OCCURRED TO TRUE.
       011000 OUTPUT-ERROR SECTION.
       011100       USE AFTER STANDARD ERROR PROCEDURE ON OUTPUT.
       011200 OUTPUT-ERROR-PARA.
  103  011300       DISPLAY OUTPUT-MESSAGE.
  104  011400       MOVE PRINT-FILE-STATUS TO STATUS-VALUE.
  105  011500       DISPLAY ERROR-DATA.
  106  011600     SET ERROR-OCCURRED TO TRUE.
       011700 END DECLARATIVES.
       011800 MAIN-PROCESSING SECTION.
       011900 MAIN-PROCEDURE.
  107  012000     MOVE "OPEN" TO OP-NAME.
  108  012100     OPEN INPUT INPUT-FILE
       012200         I-O MASTER-FILE
       012300         OUTPUT PRINT-FILE.
  109  012400     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
  111  012500     PERFORM PAGE-START.
  112  012600     PERFORM READ-INPUT-FILE.
  113  012700     PERFORM PROCESS-DATA THRU READ-INPUT-FILE
       012800                           UNTIL THE-END-OF-INPUT.
  114  012900     PERFORM PAGE-END.
  115  013000     MOVE "CLOSE" TO OP-NAME.
  116  013100     CLOSE INPUT-FILE
       013200           MASTER-FILE
       013300           PRINT-FILE.
  117  013400     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
  119  013500     STOP RUN.
       013600
       013700 PROCESS-DATA.
  120  013800     IF INPUT-DET-FLD EQUAL SPACES
  121  013900         PERFORM INIT-SEQUENTIAL-PROCESS
       014000     ELSE
  122  014100         PERFORM DYNAMIC-PROCESS.
       014200 READ-INPUT-FILE.
  123  014300     MOVE "READ" TO OP-NAME.
  124  014400     READ INPUT-FILE
  125  014500         AT END SET THE-END-OF-INPUT TO TRUE.
  126  014600     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
       014700
       014800 INIT-SEQUENTIAL-PROCESS.
  128  014900     MOVE INPUT-GEN-FLD TO MASTER-GEN-FLD.
  129  015000     MOVE "START" TO OP-NAME.
  130  015100     START MASTER-FILE
       015200          KEY IS NOT LESS THAN MASTER-GEN-FLD
       015300          INVALID KEY
```

Figure 41 (Part 3 of 5). Example of Indexed File Updating

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  131  015400              DISPLAY "MASTER-FILE START FAILED: INVALID KEY ",
       015500                      MASTER-GEN-FLD
  132  015600              MOVE HIGH-VALUE TO MASTER-GEN-FLD.
  133  015700     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
  135  015800     PERFORM SEQUENTIAL-PROCESS
       015900         UNTIL INPUT-GEN-FLD NOT EQUAL MASTER-GEN-FLD.
       016000
       016100 SEQUENTIAL-PROCESS.
  136  016200     MOVE "READ NEXT" TO OP-NAME.
  137  016300     READ MASTER-FILE NEXT RECORD
  138  016400         AT END MOVE HIGH-VALUE TO MASTER-GEN-FLD.
  139  016500     IF ERROR-OCCURRED GO TO  ERROR-TERMINATION.
  141  016600     IF INPUT-GEN-FLD EQUAL MASTER-GEN-FLD
  142  016700         MOVE MASTER-KEY TO PRINT-KEY
  143  016800         MOVE MASTER-NAME TO PRINT-NAME
  144  016900         MOVE MASTER-BAL TO PRINT-NEW-BAL
  145  017000         PERFORM PRINT-DETAIL.
       017100
       017200 DYNAMIC-PROCESS.
  146  017300     MOVE INPUT-KEY TO MASTER-KEY.
  147  017400     MOVE "READ" TO OP-NAME.
  148  017500     READ MASTER-FILE
       017600         INVALID KEY
  149  017700           DISPLAY "MASTER-FILE READ FAILED: INVALID KEY ",
       017800                   MASTER-KEY
  150  017900           MOVE HIGH-VALUE TO MASTER-GEN-FLD.
  151  018000     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
  153  018100     IF INPUT-GEN-FLD EQUAL MASTER-GEN-FLD
  154  018200         MOVE MASTER-KEY TO PRINT-KEY
  155  018300         MOVE MASTER-NAME TO PRINT-NAME
  156  018400         MOVE MASTER-BAL TO PRINT-BAL
  157  018500         MOVE INPUT-AMT TO PRINT-AMT
  158  018600         ADD INPUT-AMT TO MASTER-BAL
  159  018700         MOVE MASTER-BAL TO PRINT-NEW-BAL
  160  018800         PERFORM PRINT-DETAIL
  161  018900         MOVE "REWRITE" TO OP-NAME
  162  019000         REWRITE MASTER-RECORD
       019100            INVALID KEY
  163  019200              DISPLAY "MASTER-FILE REWRITE FAILED: INVALID KEY ",
       019300                      MASTER-KEY
  164  019400              MOVE HIGH-VALUE TO MASTER-GEN-FLD.
  165  019500     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
       019600 PRINT-DETAIL.
  167  019700     MOVE "WRITE" TO OP-NAME.
  168  019800     WRITE PRINT-RECORD-1
       019900          AT END-OF-PAGE
  169  020000          PERFORM PAGE-END THROUGH PAGE-START.
  170  020100     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
  172  020200     MOVE SPACES TO PRINT-RECORD-1.
       020300
       020400 PAGE-END.
  173  020500     MOVE "WRITE" TO OP-NAME.
  174  020600     ADD 1 TO PG-NUMBER.
  175  020700     SUBTRACT LINAGE-COUNTER OF PRINT-FILE FROM 12
       020800          GIVING LINES-TO-FOOT.
  176  020900     MOVE SPACES TO PRINT-RECORD-1.
  177  021000     WRITE PRINT-RECORD-1
       021100          AFTER ADVANCING LINES-TO-FOOT.
  178  021200     WRITE PRINT-RECORD-2 FROM PAGE-FOOT
       021300          BEFORE ADVANCING PAGE.
  179  021400     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
       021500 PAGE-START.
```

Figure 41 (Part 4 of 5). Example of Indexed File Updating

```
5763CB1                              COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  181  021600     WRITE PRINT-RECORD-2 FROM PAGE-HEAD
       021700          AFTER ADVANCING 0 LINES.
  182  021800     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
  184  021900     MOVE SPACES TO PRINT-RECORD-2.
  185  022000     WRITE PRINT-RECORD-2 FROM COLUMN-HEAD
       022100          AFTER ADVANCING 1 LINE.
  186  022200     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
  188  022300     MOVE SPACES TO PRINT-RECORD-2.
       022400 ERROR-TERMINATION.
  189  022500     DISPLAY "PROCESS TERMINATING ABNORMALLY".
  190  022600     STOP RUN.
                        * * * * *  E N D  O F  S O U R C E  * * * * *
5763CB1                                COBOL MESSAGES
 STMT
*   29  MSGID: CBL0650  SEVERITY: 00  SEQNBR:  002900
       Message . . . . :   Blocking/Deblocking for file 'INPUT-FILE'
         will be performed by compiler-generated code.
                                      MESSAGE SUMMARY
  TOTAL    INFO(0-4)    WARNING(5-19)    ERROR(20-29)    SEVERE(30-39)    TERMINAL(40-99)
     1          1              0                0               0                0
                  * * * * *  E N D  O F  C O B O L  M E S S A G E S  * * * * *
```

Figure 41 (Part 5 of 5). Example of Indexed File Updating

Relative File Creation

This program creates a relative file of summary sales records using sequential access. Each record contains a five-year summary of unit and dollar sales for one week of the year; there are 52 records within the file, each representing one week.

Each input record represents the summary sales for one week of one year. The records for the first week of the last five years (in ascending order) are the first five input records. The records for the second week of the last five years are the next five input records, and so on. Thus, five input records fill one output record.

The `RELATIVE KEY` for the `RELATIVE-FILE` is not specified because it is not required for sequential access unless the `START` statement is used.

```
5763CB1                              COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.   CREATEREL.
    3  000300
    4  000400 ENVIRONMENT DIVISION.
    5  000500 CONFIGURATION SECTION.
    6  000600 SOURCE-COMPUTER.   IBM-S38.
    7  000700 OBJECT-COMPUTER.   IBM-S38.
    8  000800 SPECIAL-NAMES.  REQUESTOR IS REQUESTOR.
    9  000900 FILE-CONTROL.
   10  001000     SELECT RELATIVE-FILE ASSIGN TO DISK-FILED
   11  001100         ORGANIZATION IS RELATIVE
   12  001200         ACCESS IS SEQUENTIAL
   13  001300         FILE STATUS RELATIVE-FILE-STATUS.
   14  001400     SELECT INPUT-FILE ASSIGN TO DISK-FILEC
   15  001500         FILE STATUS INPUT-FILE-STATUS.
   16  001600
   17  001700 DATA DIVISION.
   18  001800 FILE SECTION.
   19  001900 FD  RELATIVE-FILE LABEL RECORDS ARE STANDARD.
   20  002000 01  RELATIVE-RECORD-01.
   21  002100     05  RELATIVE-RECORD OCCURS 5 TIMES INDEXED BY REL-INDEX.
   22  002200         10  RELATIVE-YEAR          PICTURE 99.
   23  002300         10  RELATIVE-WEEK          PICTURE 99.
   24  002400         10  RELATIVE-UNIT-SALES    PICTURE S9(6).
   25  002500         10  RELATIVE-DOLLAR-SALES  PICTURE S9(9)V99.
   26  002600 FD  INPUT-FILE LABEL RECORDS STANDARD.
   27  002700 01  INPUT-RECORD.
   28  002800     05  INPUT-YEAR                PICTURE 99.
   29  002900     05  INPUT-WEEK                PICTURE 99.
   30  003000     05  INPUT-UNIT-SALES          PICTURE S9(6).
   31  003100     05  INPUT-DOLLAR-SALES        PICTURE S9(9)V99.
   32  003200 WORKING-STORAGE SECTION.
   33  003300 77  INPUT-FILE-STATUS             PICTURE XX.
   34  003400 77  RELATIVE-FILE-STATUS          PICTURE XX.
   35  003500 01  WORK-RECORD.
   36  003600     05  WORK-YEAR                 PICTURE 99 VALUE 00.
   37  003700     05  WORK-WEEK                 PICTURE 99.
   38  003800     05  WORK-UNIT-SALES           PICTURE S9(6).
   39  003900     05  WORK-DOLLAR-SALES         PICTURE S9(9)V99.
   40  004000 01  ERROR-INFO.
   41  004100     05  OP-NAME                   PICTURE X(5).
   42  004200     05  FILLER                    PICTURE X(10)
   43  004300                                     VALUE " ERROR ON ".
   44  004400     05  FILE-NAME                 PICTURE X(13).
   45  004500     05  FILLER                    PICTURE X(16)
   46  004600                                     VALUE " FILE STATUS IS ".
```

Figure 42 (Part 1 of 2). Example of Relative File Creation

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   47  004700     05  STATUS-VALUE               PICTURE XX.
   48  004800 01  ERROR-FLAG                    PICTURE X VALUE SPACE.
   49  004900     88  ERROR-OCCURRED             VALUE "1".
   50  005000 01  INPUTEND                      PICTURE X VALUE SPACE.
   51  005100     88  THE-END-OF-INPUT           VALUE "E".
   52  005200
   53  005300 PROCEDURE DIVISION.
       005400 DECLARATIVES.
       005500
       005600 INP-FILE-ERROR SECTION.
       005700           USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.
       005800 INPUT-FILE-ERROR.
   54  005900     MOVE "INPUT-FILE" TO FILE-NAME.
   55  006000     MOVE INPUT-FILE-STATUS TO STATUS-VALUE.
   56  006100     SET ERROR-OCCURRED TO TRUE.
       006200 REL-FILE-ERROR SECTION.
       006300           USE AFTER STANDARD ERROR PROCEDURE ON RELATIVE-FILE.
       006400 RELATIVE-FILE-ERROR.
   57  006500     MOVE "RELATIVE-FILE" TO FILE-NAME.
   58  006600     MOVE RELATIVE-FILE-STATUS TO STATUS-VALUE.
   59  006700     SET ERROR-OCCURRED TO TRUE.
       006800 END DECLARATIVES.
       006900 BEGIN-PROCESSING SECTION.
       007000 PROCESSING-CONTROL.
   60  007100     MOVE "OPEN" TO OP-NAME.
   61  007200     OPEN INPUT INPUT-FILE
       007300         OUTPUT RELATIVE-FILE.
   62  007400     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
   64  007500     SET REL-INDEX TO 1.
   65  007600     PERFORM READ-INPUT-FILE.
   66  007700     PERFORM PROCESS-DATA THRU READ-INPUT-FILE
       007800                          UNTIL THE-END-OF-INPUT.
   67  007900     CLOSE RELATIVE-FILE INPUT-FILE.
   68  008000     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
   70  008100     STOP RUN.
       008200 ERROR-TERMINATION.
   71  008300     DISPLAY ERROR-INFO UPON REQUESTOR.
   72  008400     DISPLAY "PROCESSING TERMINATED DUE TO I-O ERROR"
       008500             UPON REQUESTOR.
   73  008600     STOP RUN.
       008700 PROCESS-DATA.
   74  008800     MOVE INPUT-RECORD TO RELATIVE-RECORD (REL-INDEX).
   75  008900     IF REL-INDEX NOT = 5
   76  009000         SET REL-INDEX UP BY 1
       009100     ELSE
   77  009200         SET REL-INDEX TO 1
   78  009300         PERFORM RELATIVE-FILE-WRITE.
       009400 READ-INPUT-FILE.
   79  009500     MOVE "READ" TO OP-NAME.
   80  009600     READ INPUT-FILE
   81  009700         AT END SET THE-END-OF-INPUT TO TRUE.
   82  009800     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
       009900 RELATIVE-FILE-WRITE.
   84  010000     MOVE "WRITE" TO OP-NAME.
   85  010100     WRITE RELATIVE-RECORD-01.
   86  010200     IF ERROR-OCCURRED GO TO ERROR-TERMINATION.
                         * * * * *  E N D  O F  S O U R C E  * * * * *
 5763CB1                              COBOL MESSAGES
  STMT
*   19  MSGID: CBL0650  SEVERITY: 00  SEQNBR:  001900
       Message . . . . :   Blocking/Deblocking for file 'RELATIVE-FILE'
         will be performed by compiler-generated code.
*   26  MSGID: CBL0650  SEVERITY: 00  SEQNBR:  002600
       Message . . . . :   Blocking/Deblocking for file 'INPUT-FILE'
         will be performed by compiler-generated code.
                                      MESSAGE SUMMARY
  TOTAL    INFO(0-4)    WARNING(5-19)    ERROR(20-29)    SEVERE(30-39)    TERMINAL(40-99)
     2          2            0               0               0               0
                  * * * * *  E N D  O F  C O B O L  M E S S A G E S  * * * * *
```

Figure 42 (Part 2 of 2). Example of Relative File Creation

Relative File Updating

This program uses sequential access to update the file of summary sales records created in the CREATEREL program. The updating program adds a record for the new year and deletes the oldest year's records from the RELATIVE-FILE.

The input record represents the summary sales record for one week of the preceding year. The RELATIVE KEY for the RELATIVE-FILE is present in the input record as INPUT-WEEK. The RELATIVE KEY is used to check that the record was correctly written.

```
5763CB1                              COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.  UPDATEREL.
    3  000300
    4  000400 ENVIRONMENT DIVISION.
    5  000500 CONFIGURATION SECTION.
    6  000600 SOURCE-COMPUTER.   IBM-S38.
    7  000700 OBJECT-COMPUTER.   IBM-S38.
    8  000800 INPUT-OUTPUT SECTION.
    9  000900 FILE-CONTROL.
   10  001000     SELECT RELATIVE-FILE ASSIGN TO DISK-FILED
   11  001100         ORGANIZATION IS RELATIVE
   12  001200         ACCESS IS SEQUENTIAL
   13  001300         RELATIVE KEY INPUT-WEEK
   14  001400         FILE STATUS STATUS-VALUE.
   15  001500     SELECT INPUT-FILE ASSIGN TO DISK-FILES
   16  001600         FILE STATUS STATUS-VALUE.
   17  001700
   18  001800 DATA DIVISION.
   19  001900 FILE SECTION.
   20  002000 FD  RELATIVE-FILE LABEL RECORDS STANDARD.
   21  002100 01  RELATIVE-RECORD               PICTURE X(105).
   22  002200 FD  INPUT-FILE LABEL RECORDS STANDARD.
   23  002300 01  INPUT-RECORD.
   24  002400     05  INPUT-YEAR                PICTURE 99.
   25  002500     05  INPUT-WEEK                PICTURE 99.
   26  002600     05  INPUT-UNIT-SALES          PICTURE S9(6).
   27  002700     05  INPUT-DOLLAR-SALES        PICTURE S9(9)V99.
   28  002800 WORKING-STORAGE SECTION.
   29  002900
   30  003000 01  INPUTEND                      PICTURE X VALUE SPACE.
   31  003100     88  THE-END-OF-INPUT          VALUE "E".
   32  003200 01  WORK-RECORD.
   33  003300     05  FILLER                    PICTURE X(21).
   34  003400     05  CURRENT-WORK-YEARS        PICTURE X(84).
   35  003500     05  NEW-WORK-YEAR.
   36  003600     10  WORK-YEAR                 PICTURE 99.
   37  003700     10  WORK-WEEK                 PICTURE 99.
   38  003800     10  WORK-UNIT-SALES           PICTURE S9(6).
   39  003900     10  WORK-DOLLAR-SALES         PICTURE S9(9)V99.
   40  004000 66  WORK-OUT-RECORD RENAMES
   41  004100     CURRENT-WORK-YEARS THROUGH NEW-WORK-YEAR.
   42  004200 01  ERROR-MESSAGE.
   43  004300     05  OP-NAME                  PICTURE X(7).
   44  004400     05  FILLER                   PICTURE X(10)
   45  004500                                     VALUE " ERROR ON ".
   46  004600     05  FILE-NAME                PICTURE X(13).
   47  004700     05  FILLER                   PICTURE X(16)
   48  004800                                     VALUE " FILE STATUS IS ".
   49  004900     05  STATUS-VALUE             PICTURE X(2).
   50  005000
```

Figure 43 (Part 1 of 2). Example of Relative File Updating

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   51  005100 PROCEDURE DIVISION.
       005200 DECLARATIVES.
       005300 I-O-ERROR SECTION.
       005400     USE AFTER STANDARD ERROR PROCEDURE ON RELATIVE-FILE,
       005500                                          INPUT-FILE.
       005600 ERROR-PROCEDURE.
   52  005700     DISPLAY ERROR-MESSAGE.
   53  005800     DISPLAY "PROCESSING TERMINATING".
   54  005900     STOP RUN.
       006000 END DECLARATIVES.
       006100 MAIN-PROCEDURE SECTION.
       006200 BEGIN-PROCESSING.
   55  006300     MOVE "OPEN" TO OP-NAME.
   56  006400     MOVE "INPUT-FILE" TO FILE-NAME.
   57  006500     OPEN INPUT INPUT-FILE.
   58  006600     MOVE "RELATIVE-FILE" TO FILE-NAME.
   59  006700     OPEN I-O RELATIVE-FILE.
   60  006800     PERFORM READ-FILES.
   61  006900     PERFORM UPDATE-RELATIVE-FILE THRU READ-FILES
       007000                       UNTIL THE-END-OF-INPUT.
   62  007100     MOVE "CLOSE" TO OP-NAME.
   63  007200     MOVE "INPUT-FILE" TO FILE-NAME.
   64  007300     CLOSE INPUT-FILE.
   65  007400     MOVE "RELATIVE-FILE" TO FILE-NAME.
   66  007500     CLOSE RELATIVE-FILE.
   67  007600     STOP RUN.
       007700 UPDATE-RELATIVE-FILE.
   68  007800     MOVE "REWRITE" TO OP-NAME.
   69  007900     MOVE "RELATIVE-FILE" TO FILE-NAME.
   70  008000     REWRITE RELATIVE-RECORD FROM WORK-OUT-RECORD.
       008100 READ-FILES.
   71  008200     MOVE "READ" TO OP-NAME.
   72  008300     MOVE "RELATIVE-FILE" TO FILE-NAME.
   73  008400     READ RELATIVE-FILE INTO WORK-RECORD
   74  008500         AT END SET THE-END-OF-INPUT TO TRUE.
   75  008600     MOVE "INPUT-FILE" TO FILE-NAME.
   76  008700     READ INPUT-FILE INTO NEW-WORK-YEAR
   77  008800         AT END SET THE-END-OF-INPUT TO TRUE.
                        * * * * *  E N D  O F  S O U R C E  * * * * *
5763CB1                              COBOL MESSAGES
 STMT
*   22  MSGID: CBL0650  SEVERITY: 00  SEQNBR:  002200
       Message . . . . :   Blocking/Deblocking for file 'INPUT-FILE'
         will be performed by compiler-generated code.
                                     MESSAGE SUMMARY
  TOTAL    INFO(0-4)    WARNING(5-19)    ERROR(20-29)    SEVERE(30-39)    TERMINAL(40-99)
     1          1              0               0               0                0
                  * * * * *  E N D  O F  C O B O L  M E S S A G E S  * * * * *
```

Figure 43 (Part 2 of 2). Example of Relative File Updating

Relative File Retrieval

This program, using dynamic access, retrieves the summary file created by the CREATEREL program.

The records of the INPUT-FILE contain one required field (INPUT-WEEK), which is the RELATIVE KEY for RELATIVE-FILE, and one optional field (END-WEEK). An input record containing data in INPUT-WEEK and spaces in END-WEEK requests a printout for that one specific RELATIVE-RECORD; the record is retrieved through random access. An input record containing data in both INPUT-WEEK and END-WEEK requests a printout of all the RELATIVE-FILE records within the RELATIVE KEY range of INPUT-WEEK through END-WEEK, inclusive; these records are retrieved through sequential access.

```
5763CB1                              COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.  RETRIEVAL.
    3  000300
    4  000400 ENVIRONMENT DIVISION.
    5  000500 CONFIGURATION SECTION.
    6  000600 SOURCE-COMPUTER.   IBM-S38.
    7  000700 OBJECT-COMPUTER.   IBM-S38.
    8  000800 SPECIAL-NAMES.  REQUESTOR IS REQUESTOR.
    9  000900 INPUT-OUTPUT SECTION.
   10  001000 FILE-CONTROL.
   11  001100     SELECT RELATIVE-FILE ASSIGN TO DISK-FILED
   12  001200         ORGANIZATION IS RELATIVE
   13  001300         ACCESS IS DYNAMIC
   14  001400         RELATIVE KEY INPUT-WEEK
   15  001500         FILE STATUS IS RELATIVE-FILE-STATUS.
   16  001600     SELECT INPUT-FILE ASSIGN TO DISK-FILEF
   17  001700         FILE STATUS IS INPUT-FILE-STATUS.
   18  001800     SELECT PRINT-FILE ASSIGN TO PRINTER-QSYSPRT
   19  001900         FILE STATUS IS PRINT-FILE-STATUS.
   20  002000
   21  002100 DATA DIVISION.
   22  002200 FILE SECTION.
   23  002300 FD  RELATIVE-FILE LABEL RECORDS STANDARD.
   24  002400 01  RELATIVE-RECORD-01.
   25  002500     05  RELATIVE-RECORD OCCURS 5 TIMES INDEXED BY REL-INDEX.
   26  002600         10  RELATIVE-YEAR         PICTURE 99.
   27  002700         10  RELATIVE-WEEK         PICTURE 99.
   28  002800         10  RELATIVE-UNIT-SALES   PICTURE S9(6).
   29  002900         10  RELATIVE-DOLLAR-SALES PICTURE S9(9)V99.
   30  003000 FD  INPUT-FILE LABEL RECORDS STANDARD.
   31  003100 01  INPUT-RECORD.
   32  003200     05 INPUT-WEEK                 PICTURE 99.
   33  003300     05  END-WEEK                  PICTURE 99.
   34  003400 FD  PRINT-FILE LABEL RECORDS OMITTED.
   35  003500 01  PRINT-RECORD.
   36  003600     05 PRINT-WEEK                 PICTURE 99.
   37  003700     05 FILLER                     PICTURE X(5).
   38  003800     05 PRINT-YEAR                 PICTURE 99.
   39  003900     05 FILLER                     PICTURE X(5).
   40  004000     05 PRINT-UNIT-SALES           PICTURE ZZZ,ZZ9.
   41  004100     05 FILLER                     PICTURE X(5).
   42  004200     05 PRINT-DOLLAR-SALES         PICTURE $$$$,$$$,$$$.99.
```

Figure 44 (Part 1 of 3). Example of Relative File Retrieval

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   43  004300 WORKING-STORAGE SECTION.
   44  004400 77  RELATIVE-FILE-STATUS          PICTURE XX.
   45  004500 77  INPUT-FILE-STATUS             PICTURE XX.
   46  004600 77  PRINT-FILE-STATUS             PICTURE XX.
   47  004700 77  HIGH-WEEK                     PICTURE 99 VALUE 53.
   48  004800 77  OP-NAME                       PICTURE X(9).
   49  004900 01  INPUTEND                      PICTURE X(9).
   50  005000     88  THE-END-OF-INPUT          VALUE "E".
   51  005100 PROCEDURE DIVISION.
       005200 DECLARATIVES.
       005300 RELATIVE-FILE-ERROR SECTION.
       005400          USE AFTER STANDARD ERROR PROCEDURE ON RELATIVE-FILE.
       005500 RELATIVE-ERROR-MSG.
   52  005600     DISPLAY OP-NAME, " ERROR ON RELATIVE-FILE ".
   53  005700     DISPLAY "FILE STATUS VALUE IS ", RELATIVE-FILE-STATUS.
   54  005800     DISPLAY "PROCESSING TERMINATED ".
   55  005900     STOP RUN.
       006000 INPUT-FILE-ERROR SECTION.
       006100          USE AFTER STANDARD ERROR PROCEDURE ON INPUT-FILE.
       006200 INPUT-ERROR-MSG.
   56  006300     DISPLAY OP-NAME, " ERROR ON INPUT-FILE ".
   57  006400     DISPLAY "FILE STATUS VALUE IS ", INPUT-FILE-STATUS.
   58  006500     DISPLAY "PROCESSING TERMINATED ".
   59  006600     STOP RUN.
       006700 PRINT-FILE-ERROR SECTION.
       006800          USE AFTER STANDARD ERROR PROCEDURE ON PRINT-FILE.
       006900 PRINT-ERROR-MSG.
   60  007000     DISPLAY OP-NAME, " ERROR ON PRINT-FILE ".
   61  007100     DISPLAY "FILE STATUS VALUE IS ", PRINT-FILE-STATUS.
   62  007200     DISPLAY "PROCESSING TERMINATED ".
   63  007300     STOP RUN.
       007400 END DECLARATIVES.
       007500 MAIN-PROCEDURE SECTION.
       007600 MAIN-PROCESSING.
   64  007700     MOVE "OPEN" TO OP-NAME.
   65  007800     OPEN INPUT INPUT-FILE RELATIVE-FILE
       007900          OUTPUT PRINT-FILE.
   66  008000     MOVE SPACES TO PRINT-RECORD.
   67  008100     PERFORM READ-INPUT-FILE.
   68  008200     PERFORM CONTROL-PROCESS THRU READ-INPUT-FILE
       008300                          UNTIL THE-END-OF-INPUT.
   69  008400     MOVE "CLOSE" TO OP-NAME.
   70  008500     CLOSE RELATIVE-FILE
       008600          INPUT-FILE
       008700          PRINT-FILE.
   71  008800     STOP RUN.
       008900 CONTROL-PROCESS.
   72  009000     IF (END-WEEK = SPACES OR END-WEEK = 00)
   73  009100         PERFORM RANDOM-PROCESS
       009200     ELSE
   74  009300         PERFORM SEQUENTIAL-PROCESS.
       009400 READ-INPUT-FILE.
   75  009500     MOVE "READ" TO OP-NAME.
   76  009600     READ INPUT-FILE
   77  009700         AT END SET THE-END-OF-INPUT TO TRUE.
```

Figure 44 (Part 2 of 3). Example of Relative File Retrieval

```
5763CB1                              COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
       009800 RANDOM-PROCESS.
   78  009900     MOVE "READ" TO OP-NAME.
   79  010000     READ RELATIVE-FILE
   80  010100          INVALID KEY MOVE HIGH-WEEK TO RELATIVE-WEEK(1).
   81  010200     IF RELATIVE-WEEK(1) NOT EQUAL HIGH-WEEK
   82  010300          PERFORM PRINT-SUMMARY VARYING REL-INDEX FROM 1 BY 1
       010400                                UNTIL REL-INDEX > 5.
       010500 SEQUENTIAL-PROCESS.
   83  010600     MOVE "READ" TO OP-NAME.
   84  010700     READ RELATIVE-FILE
   85  010800          INVALID KEY MOVE HIGH-WEEK TO RELATIVE-WEEK(1).
   86  010900     PERFORM READ-REL-SEQ
       011000          UNTIL RELATIVE-WEEK(1) GREATER THAN END-WEEK.
       011100
       011200 READ-REL-SEQ.
   87  011300     PERFORM PRINT-SUMMARY VARYING REL-INDEX FROM 1 BY 1
       011400                           UNTIL REL-INDEX > 5.
   88  011500     MOVE "READ NEXT" TO OP-NAME.
   89  011600     READ RELATIVE-FILE NEXT RECORD
   90  011700         AT END MOVE HIGH-WEEK TO RELATIVE-WEEK(1).
       011800 PRINT-SUMMARY.
   91  011900     MOVE RELATIVE-YEAR (REL-INDEX) TO PRINT-YEAR.
   92  012000     MOVE RELATIVE-WEEK (REL-INDEX) TO PRINT-WEEK.
   93  012100     MOVE RELATIVE-UNIT-SALES (REL-INDEX) TO PRINT-UNIT-SALES.
   94  012200     MOVE RELATIVE-DOLLAR-SALES(REL-INDEX) TO PRINT-DOLLAR-SALES.
   95  012300     MOVE "WRITE" TO OP-NAME.
   96  012400     WRITE PRINT-RECORD AFTER ADVANCING 2 LINES.
                         * * * * *  E N D  O F  S O U R C E  * * * * *
 5763CB1                               COBOL MESSAGES
  STMT
*   30  MSGID: CBL0650  SEVERITY: 00  SEQNBR:  003000
       Message . . . . :   Blocking/Deblocking for file 'INPUT-FILE'
         will be performed by compiler-generated code.
                                      MESSAGE SUMMARY
  TOTAL    INFO(0-4)    WARNING(5-19)    ERROR(20-29)    SEVERE(30-39)    TERMINAL(40-99)
      1          1            0              0               0                0
                  * * * * *  E N D  O F  C O B O L  M E S S A G E S  * * * * *
```

Figure 44 (Part 3 of 3). Example of Relative File Retrieval

Chapter 7. System/38-Compatible COBOL Programming Considerations

This chapter describes:

- The device-independent and device-dependent characteristics of System/38-Compatible COBOL on the AS/400 system.
- Input and output spooling functions.
- System override considerations.
- File and record locking considerations.
- Unblocking and blocking records to improve performance.
- General information about the use of externally described files and program described files in the System/38-Compatible COBOL program.
- Format 2 `COPY` statement, (DDS or DD Formats).
- System/38-Compatible COBOL functions that relate specifically to COBOL `PRINTER` devices.
- Commitment control considerations.
- Performance considerations.
- Recovery after a failure.
- Inter-Program Communications considerations.
- General information about the local data area available to a COBOL program.
- File considerations.

You might need to refer to other AS/400 manuals for information about a particular topic in this chapter. They are listed below:

- *CallPath/400 Planning and Installation Guide*, GA21-9601, SC21-9601, which provides the following information:
  - Communications information that is common among AS/400 communications support, such as:
    - Setting and changing communication values
    - Starting and stopping communications
  - Communication configuration information, such as defining lines, controllers, and devices
  - Information about defining and using display station pass-through
  - Information about the 3270 remote attachment.
- *Database Guide*, which contains a detailed discussion of the AS/400 system data base structure. This manual also describes how to define files to the system using data description specifications (DDS) keywords.
- *Data Management Guide*, which contains information about overriding and copying files, describing display, printer, tape, and diskette files to the system, as well as spooling and output queues.

In addition, you might need to refer to the following System/38 publications for information about a particular topic in this chapter which would pertain to the AS/400 System/38 environment. They are listed below:

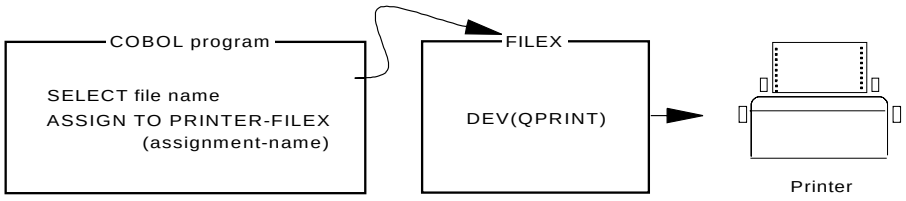
- *IBM System/38 Control Program Facility Programmer's Guide*, SC21-7730, which explains how to use CPF commands and data description specifications.
- *IBM System/38 Data Communications Programmer's Guide*, SC21-7825, which described commands, parameters, and data description specification keywords that are used for program-to-program and system-to-device communication functions.

Device Independence/Device Dependence

The key element for all I-O operations on the AS/400 system is the file. All files used on the system are defined to OS/400. OS/400 maintains a description of each file that is accessed by a program when the file is used.

The files are kept online and serve as the connecting link between a program and the device used for I-O. The actual device association is made when the file is processed. In some instances, this type of I-O control allows the user to change the attribute of the file (and, in some cases, change the device) used in a program without changing the program.

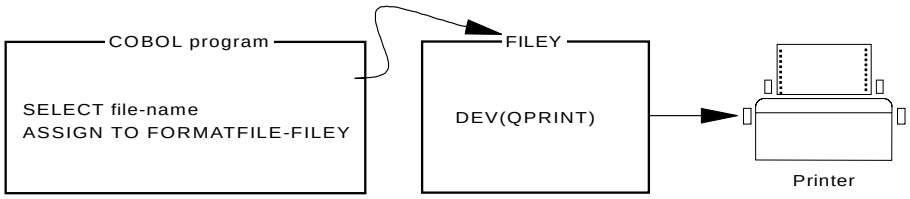
In System/38-Compatible COBOL, the file name specified in the `assignment-name` entry of the `ASSIGN` clause of the file control entry is used to point to the file. This file name points to the OS/400 file description:



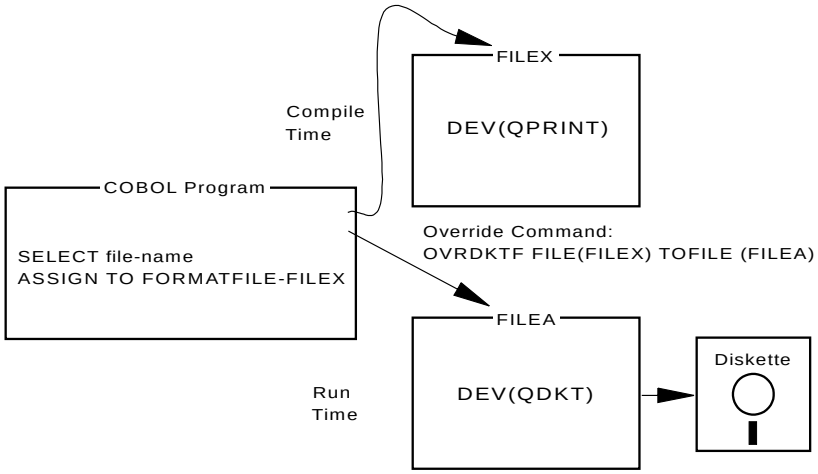
The COBOL device name in the `ASSIGN` clause defines the COBOL functions that can be processed on the selected file. At compilation time, certain COBOL functions are valid only for a specific COBOL device name; therefore, in this respect, COBOL is device dependent. The following are examples of device dependency:

- `SUBFILE` operations are valid only for a `WORKSTATION` device.
- Indicators are valid only for `WORKSTATION` or `FORMATFILE` devices.
- `LINAGE` is valid only for the `PRINTER` device.
- `OPEN INPUT WITH NO REWIND` is valid only for a `TAPEFILE` device.

For example, assume that the file name `FILEY` is associated in the COBOL program with the `FORMATFILE` device. The device `FORMATFILE` is an independent device type; therefore, no line or page control specifications are valid in the COBOL program in the `WRITE ADVANCING` statement. When the program is run, the actual I-O device is specified in the description of `FILEY`; for example, the device might be a printer, in which case only the default line and page control or those defined in the DDS would be used:



CL commands can be used to override a parameter in the specified file description or to redirect a file at compilation time or run time. File redirection allows the user to specify one file at compilation time and another file at run time:



In the preceding example, the Override to Diskette File command (OVRDKTF) allows the program to run with an entirely different device file than was specified at compilation time.

Not all file redirections or overrides are valid. At run time, checking occurs to ensure that the specifications within the COBOL program are valid for the file being processed. OS/400 allows some file redirections even if device specifics are contained in the program. For example, if the COBOL device name is PRINTER and the actual file the program uses is not a printer, OS/400 ignores the COBOL print spacing and skipping specifications.

There are other file redirections that OS/400 does not allow and that cause program termination. For example, if the COBOL device name is DATABASE or DISK and a keyed READ operation is specified in the program, the program is terminated if the actual file the program uses is not a disk or data base file.

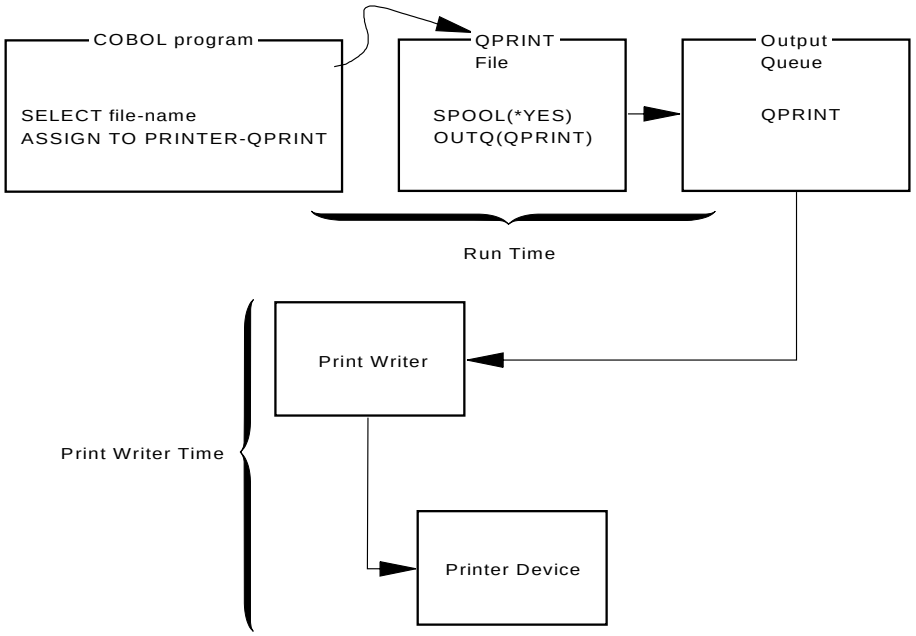
See “System Override Considerations” on page 209 for more detailed information on valid file redirections and file overrides.

Spooling

The AS/400 system provides for the use of input and output spooling functions. Each AS/400 file description contains a spool attribute that determines whether spooling is used for the file at run time. The COBOL program is not aware that spooling is being used. The actual physical device from which a file is read or to which a file is written is determined by the spool reader or the spool writer.

Output Spool

Output spooling is valid for batch and interactive jobs. The description of the file that is specified in COBOL by the system-name contains the specification for spooling as shown in the following example:

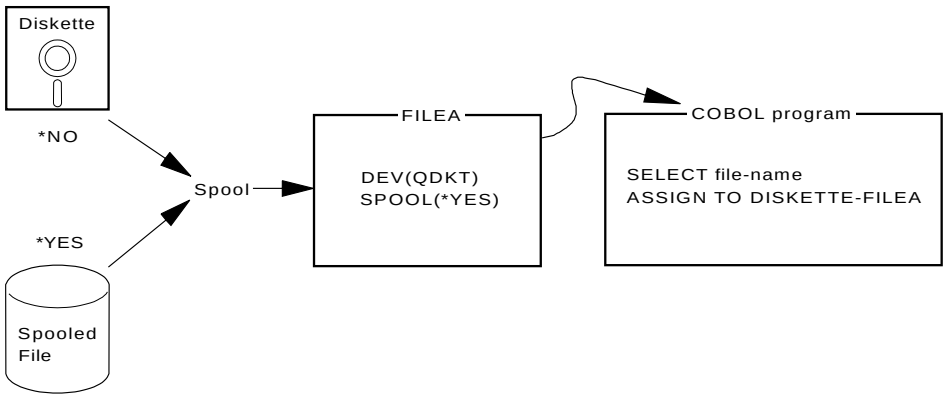


File override commands can be used at run time to override the spooling options that are specified in the file description, such as the number of copies to be printed. In addition, AS/400 spooling support allows a user to redirect a file after the program has run. For example, the user can direct the printer output to a different device, such as a diskette.

Input Spool

Input spooling is valid only for inline data files in batch jobs. If the input data read by COBOL comes from a spooled file, COBOL is not aware of which device the data was spooled in from.

The data is read from a spooled inline file:



System Override Considerations

Any overrides must be specified before the file is opened by the COBOL program. The system uses the file override command to determine the file to open and the attributes of the file.

The simplest form of overriding a file is to override some attributes of the file. For example, FILE(OUTPUT) with COPIES(2) is specified when a printer file is created. Then, before the COBOL program is run, the number of printed copies of output can be changed to 3. The override command is as follows:

```
OVRPRTF  FILE(OUTPUT)  COPIES(3)
```

Another form of file overriding is to redirect the COBOL program to access a different file. When the override redirects the program to a file of the same type (such as a printer file to another printer file), the file is processed in the same manner as the original file.

When the override redirects the program to a file of a different type, the overriding file is processed in the same manner as the original file would have been processed. However, device dependent specifications in the COBOL program are ignored and the defaults are taken by the system.

Not all file redirections are valid. For example, an indexed file for a COBOL program can only be overridden to another indexed file with a keyed access path.

Note: In particular, associated card files cannot be redirected because the compiler uses a single operation (PUTGET) to punch one record and read the next. This operation is valid only to a card device capable of both reading and punching.

Multiple member processing can be accomplished for a data base file, by overriding a data base file to process all members. You should note the following exceptions:

- A data base source file used for a COBOL program, cannot be overridden to process all members. Specifying OVRDBF MBR(*ALL) will result in the termination of the compile.
- A data base file used for a COPY statement, cannot be overridden to process all members. Specifying OVRDBF MBR(*ALL) will cause the COPY statement to be ignored.

It is the COBOL programmer's responsibility to ensure that file overrides are applied properly.

File and Record Locking by COBOL

OS/400 allows a lock state (exclusive, exclusive allow read, shared for update, shared no update, or shared for read) to be placed on a file used during a job. The file can be allocated in such a manner with the Allocate Object (ALCOBJ) command.

The ALCOBJ command can be used to specify the desired lock state. If no ALCOBJ command is used for a job, OS/400 places the following lock states on data base files when it opens the them:

OPEN Type	Lock State
INPUT	Shared-for-read
I-O	Shared-for-update
EXTEND	Shared-for-update
OUTPUT	Shared-for-update

The shared-for-read lock state allows another user to open the file with a lock state of shared-for-read, shared-for-update, shared-no-update, or exclusive-allow-read, but the user cannot specify the exclusive use of the file. The shared-for-update lock state allows another user to open the file with shared-for-read or shared-for-update lock state.

In order for programs to share a data base file, the file should be opened by the first program (the program with the highest call level in the stack) in a way that will allow subsequent programs to share the same file. If a subsequent program requests a function that was excluded by the first program, an abnormal termination occurs.

OS/400 places a shared-for-read on the device file and an exclusive lock state on the device. Another user can open the device file, but cannot use the same physical device.

The lock state placed on the file by OS/400 can be changed if you use the ALCOBJ command.

Note: When a COBOL program opens a physical file for OUTPUT, that file will be subject to an exclusive lock for the period of time necessary to clear the member. For more information see the “OPEN Statement” on page 389.

Releasing a Record Read for Update

When a data base record is read for update (i.e., the file was opened for I-O), a lock is placed on that record. If it is a logical file, then the lock is placed on the relevant records in the physical file(s) on which the logical file is based. The lock applies not only to other programs, but also to the original program if it attempts to update the same underlying physical record through a second file. COBOL releases the record from its locked state when the next successful I-O operation occurs. No special action is required to release a record from its locked state if the record does not require any changes. If a requested record is already locked by another program, a file status of 9D is returned.

Note: When a file with indexed or relative organization is opened for I-O, using random or dynamic access, a failed I-O operation on any of the I-O verbs except WRITE will also unlock the record.

Unblocking Input Records and Blocking Output Records

To potentially improve the performance of input and output operations, the COBOL compiler generates code to unblock input records and block output records if all of the following conditions exist:

- ACCESS IS SEQUENTIAL is specified for the file.
- The file is opened only for INPUT or OUTPUT in that program.
- The file is assigned to DISK, DATABASE, DISKETTE, or TAPEFILE.

- No START statements are specified for the file.

Even when all of the above conditions are met, certain OS/400 restrictions can cause blocking and unblocking to not be processed. In these cases, performance improvements will not be realized.

The I-O-FEEDBACK area is not updated after each read or write for files in which multiple records are blocked and unblocked by COBOL. See "I-O-FEEDBACK" on page 552 for more information.

For data base files, you may not see all changes as they occur, if the changes are made in different programs.

Externally Described/Program Described Files

All files on the AS/400 system are defined to OS/400. However, the extent to which files can be defined differs:

- An *externally described file* is described at the field level to OS/400 through DDS. The description includes information about the type of file, such as data base or a device, and a description of each field and its attributes.
- A *program described file* is described at the field level within the COBOL program in the Data Division. The description of the file to OS/400 includes information about the type of file and the length of the records in the file.

Both externally described files and program described files must be defined in the COBOL program within the INPUT-OUTPUT SECTION and the FILE SECTION. However, record descriptions in the FILE SECTION for externally described files can be defined with the Format 2 COPY statement.

Device-dependent functions such as forms control are not extracted by the Format 2 COPY operation. Only field level descriptions are extracted.

When EXTERNALLY-DESCRIBED-KEY is specified as RECORD KEY, the field(s) that compose RECORD KEY are also extracted from DDS.

(For more information on the Format 2 COPY statement, see Figure 51 on page 219 and the accompanying text).

Note: Actual file processing within the Procedure Division is the same, whether the file is externally described or program described.

Externally described files offer the following advantages:

- Less coding in COBOL programs. If the same file is used by many programs, the fields can be defined once to OS/400 and used by all the programs. This eliminates the need to code record descriptions for COBOL programs that use externally described files.
- Less maintenance activity when the file's record format is changed. The user can often update programs by changing the file's record format and then recompiling the programs that use the file without changing any coding in the program.
- Improved documentation because programs using the same files use consistent record format and field names.

- Any editing that is to be processed on externally described output files is specified in DDS.

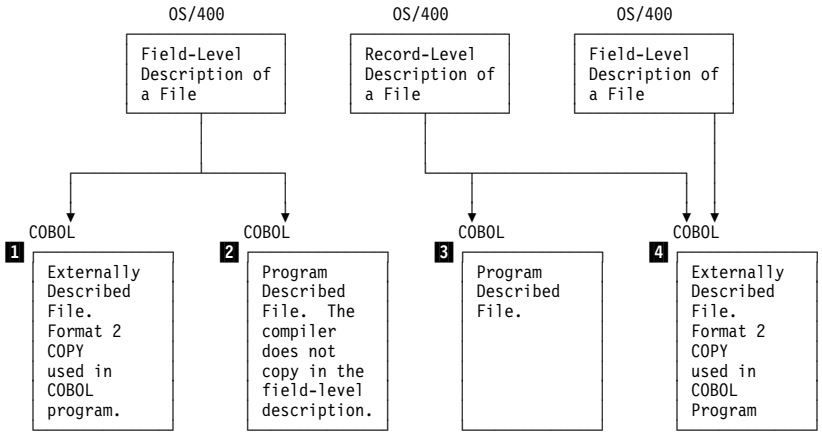
The external description for a file includes:

- The record format specifications that contain a description of the fields in a record
- Access path specifications that describe how the records are to be retrieved.

These specifications result from the DDS for the file and the OS/400 create file command that is used for the file.

If the user chooses, he can use an externally described file within the program by specifying the file as program described (specifying the coding for the record description in the source). In this case, the compiler does not copy in the external field-level description of the file at compilation time. This approach can be used in conversion where existing programs use program described files and new programs use externally described files to refer to the same file.

Figure 45 on page 213 shows some typical relationships between COBOL programs and files on the AS/400 system.



1 The COBOL program uses the field level description of a file that is defined to OS/400. The COBOL user coded a Format 2 COPY statement for his record description. At compilation time, the compiler copies in the external field-level description and translates it into a syntactically correct COBOL record description. The file must exist at compilation time.

2 An externally described file is used as a program described file in the COBOL program. The entire record description for the file is coded in the COBOL program. This file does not have to exist at compilation time.

3 A file is described to OS/400 only to the record level. The entire record description must be coded in the COBOL program. This file does not have to exist at compilation time.

4 A file-name can be specified for compilation time, and a different file-name can be specified for run time. A COBOL Format 2 COPY statement generates the record description for the file at compilation time. At run time, a different library list or a file override command can be used so that a different file is accessed by the program. The file description copied in at compilation time is used to describe the input records used at run time.

Note: For externally described files, the two file formats must be the same. Otherwise, a level check error will occur.

Figure 45. Typical Relationships between COBOL and the Files on the AS/400 system

Data Description Specifications (DDS) are used to describe files at the field level to OS/400. Each record format in an externally described file is identified by a unique record format name.

The record format specifications describe the fields in a record and the location of the fields in a record. The fields are located in the record in the order specified in DDS. The field description generally includes the field name, the field type (character, binary, zoned decimal, or packed decimal), and the field length (including the number of decimal positions in a numeric field). Instead of being specified in the record format for a physical or logical file, the field attributes can be defined in a field reference file (see Figure 46 on page 215).

The keys for a record format are specified in DDS. When you use a Format 2 `COPY` statement, a table of comments is generated in the source program listing showing how the keys for the format are defined in DDS.

In addition, DDS keywords can be used to:

- Specify edit codes for a field (`EDTCDE`)
- Specify edit words for a field (`EDTWRD`)
- Specify that duplicate key values are not allowed for the file (`UNIQUE`)
- Specify a text description for a record format or a field (`TEXT`).

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7764-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								
Programmer	Date		Key								

Description	Page of

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name Not (N)	Condition Name Indicator	Condition Name Not (N)	Condition Name Indicator	Condition Name Not (N)	Condition Name Indicator	Name Type (b/N/F/S/O)	Reserved	Name	Reference (R)	Length	Data Type (b/A/P/S/B/A/S/X/Y/N/I/U/W)	Decimal Positions	Usage (B/O/I/S/B/H/M)	Location: Line	Location: Pos	Functions
	A	*	*	FLDREF							DSTREF		DISTRIBUTION APPLICATION FIELD REFERENCE						
	A									R	DSTREF								TEXT('DISTRIBUTION FIELD REF')
	A	*	COMMON FIELDS USED AS REFERENCE																
1	A										BASDAT		6		0				EDTCDE(Y)
	A																		TEXT('BASE DATA FIELD')
	A	*	FIELDS USED BY CUSTOMER MASTER FILE																
2	A										CUST		5						CHECK(MF)
	A																		COLHDG('CUSTOMER' 'NUMBER')
	A										NAME		20						COLHDG('CUSTOMER NAME')
3	A										ADDR	R							REFFLD(NAME)
	A																		COLHDG('CUSTOMER ADDRESS')
	A										CITY	R							REFFLD(NAME)
	A																		COLHDG('CUSTOMER CITY')
2	A										STATE		2						CHECK(MF)
	A																		COLHDG('STATE')
	A										SRHCOD		6						CHECK(MF)
	A																		COLHDG('SEARCH' 'CODE')
	A																		TEXT('CUSTOMER NUMBER SEARCH CODE')
2	A										ZIP		5		0				CHECK(MF)
	A																		COLHDG('ZIP' 'CODE')
4	A										CUSTYP		1		0				RANGE(1 5)
	A																		COLHDG('CUST' 'TYPE')
	A																		TEXT('CUSTOMER TYPE 1=GOV 2=SCH 3=B+
	A																		US 4=PT 5=OTH')
5	A										ARBAL		8		2				COLHDG('ACCTS REC' 'BALANCE')
	A																		EDTCDE(J)
6	A										ORDBAL	R							REFFLD(ARBAL)
	A																		COLHDG('A/R AMT IN' 'ORDER FILE')
	A										LSTAMT	R							REFFLD(ARBAL)
	A																		COLHDG('LAST' 'AMOUNT' 'PAID')
7	A																		TEXT('LAST AMOUNT PAID IN A/R')
	A																		
	A																		
	A																		
	A																		

Figure 46 (Part 1 of 2). Example of a Field Reference File

This example of a field reference file shows the definitions of the fields that are used by the CUSMSTL (customer master logical) file. The field reference file normally contains the definitions of fields that are used by other files. The following text describes some of the entries for this field reference file.

1 The BASDAT field is edited by the Y edit code, as indicated by the keyword EDTCDE (Y). If this field is used in an externally described output file for a COBOL program, the COBOL-generated field is compatible with the data type specified in the DDS. The field is edited when the record is written. When the field is used in a program described output file, compatibility with the DDS fields in the file is the user's responsibility. When DDS is not used to create the file, appropriate editing of the field in the COBOL program is the user's responsibility.

2 The CHECK(MF) entry specifies that the field is a mandatory fill field when it is entered from a display work station. Mandatory fill means that all characters for the field must be entered from the display work station.

3 The ADDR and CITY fields share the same attributes that are specified for the NAME field, as indicated by the REFFLD keyword.

4 The RANGE keyword, which is specified for the CUSTYP field, ensures that the only valid numbers that can be entered into this field from a display work station are 1 through 5.

5 The COLHDG keyword provides a column head for the field if it is used by the Interactive Data Base Utilities (IDU).

6 The ARBAL field is edited by the J edit code, as indicated by the keyword EDTCDE(J).

7 A text description (TEXT keyword) is provided for some fields. The TEXT keyword is used for documentation purposes and appears in various listings.

Figure 46 (Part 2 of 2). Example of a Field Reference File

COBOL Specifications for Externally Described Files

The COBOL user can incorporate the file description in his program by coding a Format 2 COPY statement. The information from the external description is then retrieved by the COBOL compiler, and a COBOL data structure is generated.

The following pages provide examples of DDS usage and the COBOL code which would result from the use of a Format 2 COPY statement. (See "Format 2 COPY Statement, DDS or DD Formats" on page 219 for a detailed description of the Format 2 COPY statement).

- Figure 47 on page 217 shows the DDS for a logical file and Figure 48 on page 218 shows the COBOL code generated.
- Figure 49 on page 218 describes the same file but includes the ALIAS keyword, and Figure 50 on page 219 shows the COBOL code generated.

Actual file processing within the Procedure Division is the same for both program described and externally described files.

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7764-1 UM/080*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Condition Name: Not (N)	Condition Name: Indicator	Condition Name: Not (N)	Condition Name: Indicator	Condition Name: Not (N)	Condition Name: Indicator	Name Type (R/I/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (B/P/S/B A/E/F/X/Y/N/I/U/W)	Decimal Positions	Usage (B/O/I/D/N/M)	Location: Line	Location: Pos	Functions
1 2 3 4 5	6	7	8 9 10	11 12	13 14	15 16			17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36	37 38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
1	A	*	* L O G I C			A L					C U S M S T L		C U S T O M	E R			M A S	T E R	F I L E
	A																	2	U N I Q U E
	A	*				3	R				C U S R E C								P F I L E (C U S M S T P)
	A																		T E X T (' C U S T O M E R M A S T E R R E C O R D ')
	A										C U S T								
	A										N A M E								
	A										A D D R								
	A										C I T Y								
	A										S T A T E								
	A										Z I P								
	A										S R H C O D								
	A										C U S T Y P								
	A										A R B A L								
	A										O R D B A L								
	A										L S T A M T								
	A										L S T D A T								
	A										C R D L M T								
	A										S L S Y R		5						
	A										S L S L Y R								
	A					4	K				C U S T								

1 A logical file for processing the customer master physical file (CUSMSTP) is defined and named CUSMSTL.

2 The UNIQUE keyword indicates that duplicate key values for this file are not allowed.

3 One record format (CUSREC) is defined for the CUSMSTL file, which is to be based upon the physical file CUSMSTP.

4 The CUST field is identified as the key field for this file.

5 If field attributes (such as length, data type, and decimal positions) are not specified in the DDS for a logical file, the attributes are obtained from the corresponding field in the physical file. Any field attributes specified in the DDS for the logical file override the attributes for the corresponding field in the physical file. The definition of the fields in the physical file could refer to a field reference file. A field reference file is a data description file consisting of field names and their definitions, such as size and type. When a field reference file is used, the same fields that are used in multiple record formats have to be defined only once in the field reference file.

Figure 47. Example of Data Description Specifications

```
 01  CUS-MASTER.
     COPY DDS-CUSREC OF CUSTMAST-CUSLIB.
*I-O FORMAT: CUSREC FROM FILE CUSTMAST OF LIBRARY CUSLIB       CUSREC
*                   CUSTOMER MASTER RECORD                     CUSREC
*THE KEY DEFINITIONS FOR THE RECORD FORMAT CUSREC              CUSREC
*NUMBER   NAME    RETRIEVAL   TYPE    ALTSEQ                  CUSREC
*0001     CUST    ASCENDING   AN      NO                      CUSREC
    05   CUSREC.                                              CUSREC
        06 CUST     PIC X(5).                                 CUSREC
*               CUSTOMER NUMBER                               CUSREC
        06 NAME     PIC X(20).                                CUSREC
*               CUSTOMER NAME                                 CUSREC
        06 ADDR     PIC X(20).                                CUSREC
*               CUSTOMER ADDRESS                              CUSREC
        06 CITY     PIC X(20).                                CUSREC
*               CUSTOMER CITY                                 CUSREC
        06 STATE    PIC X(2).                                 CUSREC
*               STATE ABBREVIATION                            CUSREC
        06 ZIP      PIC S9(5)     COMP-3.                     CUSREC
*               ZIP CODE                                      CUSREC
        06 SRHCOD   PIC X(6).                                 CUSREC
*               CUSTOMER NAME SEARCH CODE                     CUSREC
        06 CUSTYP   PIC 9(1).                                 CUSREC
*               CUSTOMER TYPE                                 CUSREC
        06 ARBAL    PIC S9(6)V9(2)     COMP-3.                CUSREC
*               ACCT/REC BALANCE                              CUSREC
```

Figure 48. Example of the Results of the Format 2 COPY Statement (DDS)

Figure 46 on page 215 shows an example of the field reference file that defines the attributes for the fields used in the data base file.

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7764-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page	of
Programmer	Date		Key										

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Not (N)	Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (b/R/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (b A/P/B/S/B A/N/X/Y/N/I/M/W)	Decimal Positions	Usage (b/C/I/B/N/M)	Location: Line	Pos	Functions
	A	*	*	L	O	G	I	C	A	L			CUSMSTL			CUSTOMER			MASTER FILE
	A																		UNIQUE
	A	*								R	CUSREC								PFILE(CUSMSTP)
	A																		TEXT('CUSTOMER MASTER RECORD')
	A										CUST								ALIAS(CUSTOMER_NUMBER)
	A										NAME							1	ALIAS(CUSTOMER_NAME)
	A										ADDR								ALIAS(ADDRESS)
	A										CITY								
	A										STATE								
	A										ZIP								
	A										SRHCOD								ALIAS(SEARCH_CODE)
	A										CUSTYP								ALIAS(CUSTOMER_TYPE)
	A										ARBAL								ALIAS(ACCT_REC_BALANCE)
	A										ORDBAL								
	A										LSTAMT								
	A										LSTDAT								
	A										CRDLMT								
	A										SLSYR								
	A										SLSLYR								
	A									K	CUST								

1 The name associated with the Alias keyword, which will be included in the program.

Figure 49. Example of Data Description Specifications with ALIAS

```
 01  CUS-MASTER.
     COPY DD-CUSREC OF CUSTMAST-CUSLIB.
*I-O FORMAT: CUSREC FROM FILE CUSTMAST OF LIBRARY CUSLIB       CUSREC
*                  CUSTOMER MASTER RECORD                     CUSREC
*THE KEY DEFINITIONS FOR THE RECORD FORMAT CUSREC              CUSREC
*NUMBER   NAME              RETRIEVAL  TYPE          ALTSEQ    CUSREC
*0001     CUSTOMER-NUMBER   ASCENDING  AN            NO        CUSREC
                                                               CUSREC
    05  CUSREC.                                                CUSREC
        06 CUSTOMER-NUMBER  PIC X(5).                          CUSREC
*               CUSTOMER NUMBER                                CUSREC
        06 CUSTOMER-NAME    PIC X(20).                         CUSREC
*               CUSTOMER NAME                                  CUSREC
        06 ADDRESS          PIC X(20).                         CUSREC
*               CUSTOMER ADDRESS                               CUSREC
        06 CITY             PIC X(20).                         CUSREC
*               CUSTOMER CITY                                  CUSREC
        06 STATE            PIC X(2).                          CUSREC
*               STATE ABBREVIATION                             CUSREC
        06 ZIP              PIC S9(5)    COMP-3.               CUSREC
*               ZIP CODE                                       CUSREC
        06 SEARCH-CODE      PIC X(6).                          CUSREC
*               CUSTOMER NAME SEARCH CODE                      CUSREC
        06 CUSTOMER-TYPE    PIC 9(1)                           CUSREC
*               CUSTOMER TYPE                                  CUSREC
        06 ACCT-REC-BALANCE PIC S9(6)V9(2)   COMP-3.           CUSREC
*               ACCT/REC BALANCE                               CUSREC
```

Figure 50. Example of the Results of the Format 2 COPY Statement (DD) with the Alias Keyword

IBM Extension

Format 2 COPY Statement, DDS or DD Formats

Format 2

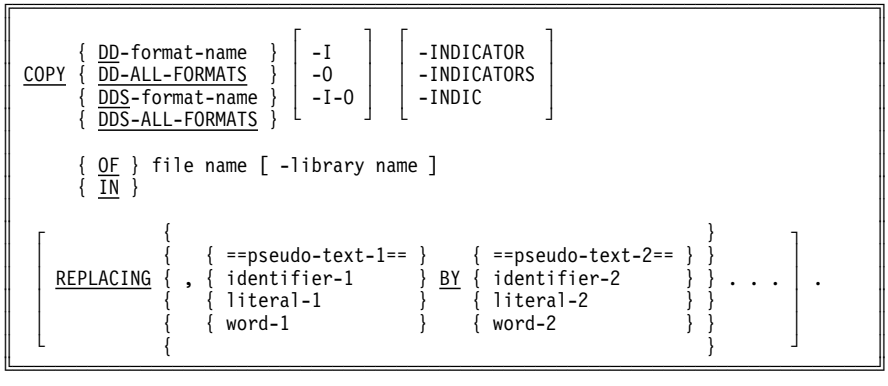


Figure 51. Format 2 COPY Statement

The Format 2 COPY statement (DDS or DD option) can be used to create COBOL Data Division statements to describe a file that exists on the system. These descriptions are based on the version of the file in existence at compile time. They

do not make use of the DDS source statements for the file. (Refer to “COPY Statement” on page 30 for general rules on the COPY statement).

The Format 2 COPY statement can be used only in the Data Division, and it is the user's responsibility to precede the statement with a group level item that has a level-number less than 05.

The DDS option copies in the internal DDS format field names.

The DD option is used to reference Alias (alternate) names. The specification of an Alias name in DDS allows a data name of up to 30 characters to be included in the COBOL program.

When the DD option is used, any Alias names present replace the corresponding DDS field names. All underscores in the Alias names are translated into hyphens before any replacing occurs.

When the RECORD KEY clause specifies EXTERNALLY-DESCRIBED-KEY, a format can be copied only once under an FD. For example, if all of the formats of a file are copied under an FD, no other Format 2 COPY statement, can be specified for the same file under that FD.

The format-name is the name of the DDS record format definition that is to be translated into COBOL data description entries. The format-name must follow the rules for formation of any System/38 environment name.

If neither -I nor -O is specified, -I-O is assumed.

If DDS-ALL-FORMATS or DD-ALL-FORMATS is specified, each record format is generated as a redefinition of an 05 elementary item defined as either:

- the size of the largest record format in the file, if the COPY statement appears in the FILE SECTION.
- the size of the largest record format that will be generated, if the COPY statement appears outside of the FILE SECTION.

If format-name is specified and both -I -O formats are to be generated, each record format is generated as a redefinition of an 05 elementary item defined as:

- the size of the largest record format that will be generated.

File name is the name of an AS/400 system file. The generated DDS entries represent the record format defined in the file. The file must be created before the program is compiled.

Library name is optional. If it is not specified, the current job library list is used as the default value.

If the file is a data base file, a single I-O format is generated.

For all other file types the description generated varies as follows:

- If -I is specified, the generated data description entries contain either:
  - The input and input/output fields for a nonsubfile format
  - The input, output, and input/output fields for a subfile format.

- If -O is specified, the generated data description entries contain either:
  - The output and input/output fields for a nonsubfile format
  - The input, output, and input/output fields for a subfile format.

Note: Subfile records with only output or input/output fields, and no field indicators specified, generate I-O formats.

The use of the INDICATOR attribute is discussed under "INDICATOR Attribute of the Format 2 COPY Statement" on page 223.

Data base files never have indicators.

If a separate storage area is needed in WORKING-STORAGE for each format, an individual COPY statement must be specified for each format.

For example, if we assume that the file CUSTMASTER contains two formats: CUSADR and CUSTDETL ; then the following COPY statements could be specified.

```
    SELECT FILE-X
    ASSIGN TO DATABASE-CUSTMASTER.
    .
    .
    .
FD  FILE-X
    LABEL RECORDS ARE STANDARD.
01  FILE-X-RECS.
    COPY DDS-ALL-FORMATS OF
       CUSTMASTER-QGPL.  (See Note 1.)
    .
    .
    .
WORKING-STORAGE SECTION.
01  ADR-REC.
    COPY DDS-CUSTADR OF
       CUSTMASTER.    (See Note 2.)
01  DETAIL-REC.
    COPY DDS-CUSTDETL OF
       CUSTMASTER.    (See Note 2.)
```

Notes:

1. This COPY statement generates only one storage area for all formats.
2. These COPY statements generate separate storage areas.

Data Structures Generated

Format (Record) Level Structures

At the beginning of each format, a table of comments is generated in the source program listing. These comments provide details of the files used during compilation of the program. If there are record keys for the file, comments are also generated to show how the keys are defined in DDS. The entries that may appear in the table and the table heading are listed below.

Heading	Possible Entry
NUMBER	key field number
NAME	key field name
RETRIEVAL	ASCENDING, DESCENDING
TYPE	ZONE, DIGIT, SIGNED, ABSVAL, AN (alphanumeric), N (numeric)
ALTSEQ	NO, YES

If redefinition is required to allow for the generation of multiple formats, a group level name is generated as follows:

```
05 file-name-RECORD
   PIC X(size of largest record).
```

for each format a group level name is assigned as follows:

- INPUT

  05 format-name-I

- OUTPUT

  05 format-name-O

- I-O Format

  05 format-name

Data Field Structures

Field names, PICTURE definitions, and numeric usage clauses are derived directly from the internal DDS format field names (or Alias names in the case of the DD option) and data type representations. Field names and PICTURE definitions are constructed as follows:

```
06  field-name PIC  (See Note 1.)
```

Notes:

1. See Figure 52 on page 223 for the appropriate COBOL definition.

DDS		COBOL DATA DIVISION n = total field length (DDS pos. 30-34) m = number of decimals (DDS pos. 36 & 37)	
Data Type (POS. 35)	Formats	If DDS pos. 36 & 37 are blank	If DDS pos. 36 & 37 are not blank
PHYSICAL, LOGICAL, PRINTER, COMMUNICATIONS, AND BSC FILES			
b (Blank)	Default	PIC X(n)	PIC S9(n-m)V9(m)
P	Packed decimal	PIC S9(n) COMP-3	PIC S9(n-m)V9(m) COMP-3
S	Zoned decimal/signed numeric	PIC S9(n)	PIC S9(n-m)V9(m)
B	Binary	PIC S9(n) COMP-4	PIC S9(n-m)V9(m) COMP-4
F	Floating Point[1]		
	- single precision	PIC S9(5) COMP-4	-
	- double precision	PIC S9(10) COMP-4	-
A	Character	PIC X(n)	-
DISPLAY FILES			
X	Alphabetic Only	PIC X(n)	-
N	Numeric Shift	PIC X(n)	PIC S9(n-m)V9(m)
Y	Numeric Only	PIC S9(n)	PIC S9(n-m)V9(m)
I	Inhibit Keyboard entry	PIC X(n)	PIC S9(n-m)V9(m)
W	Katakana	PIC X(n)	-
A	Alphanumeric Shift	PIC X(n)	-
[1] COBOL treats floating point fields as FILLER. See 'Floating Point Fields'.			

Figure 52. Data Field Structures

Indicator Structures

If indicators are requested, and exist in the format, an additional group name (06 level) is generated at the beginning of the structure, followed by entries (07 level) for the relevant individual indicators.

```
06  format-name-(I or O)-INDIC.
    07  INxx PIC 1 INDIC xx.
```

where xx is the indicator number.

For example:

```
06  SAMPLE1-I-INDIC.
    07  IN01 PIC 1 INDIC 01.
    07  IN04 PIC 1 INDIC 04.
    07  IN05 PIC 1 INDIC 05.
    07  IN07 PIC 1 INDIC 07.
06  FLD1 PIC  ...  .
06  FLD2 PIC  ...  .
```

INDICATOR Attribute of the Format 2 COPY Statement

The INDICATOR attribute specifies whether or not data description entries are generated for indicators.

If the INDICATOR attribute is specified, data description entries are generated for indicators, but not for data fields. The data description entries that are generated are determined by which one of the usage attributes (I, O, or I-O) is specified or assumed in the COPY statement.

- If ...I-INDICATOR... is specified, data description entries for input (response) indicators are generated for indicators used in the input record area.

- If ...O-INDICATOR... is specified, data description entries for output (option) indicators are generated for indicators used in the output record area.
- If ...I-O-INDICATOR... is specified or assumed, separate data description entries for both input and output (response and option) indicators are generated for indicators used in the input and output record areas.

If the INDICATOR attribute is not specified, whether data description entries are generated for indicators depends on whether the file had the keyword INDARA specified in the DDS at the time it was created.

- If INDARA was not specified, data description entries are generated for both data fields and indicators.
- If INDARA was specified, data description entries are generated for data fields only, not for indicators.

Generation of I-O Formats

When all field descriptions are identical, and the user has requested INPUT and OUTPUT fields implicitly or explicitly, only one set of field descriptions is generated. This type of description is annotated with a comment-line reading "I-O FORMAT: format-name" and neither -I nor -O is appended to the record format name.

This is always the case for data base files.

For example:

```
 01 RCUSREC.
    COPY DDS-CUSREC-I OF CUSFILE.
*   I-O FORMAT: CUSREC FROM FILE CUSFILE OF LIBRARY CUSLIB     CUSREC
* THE KEY DEFINITIONS FOR RECORD FORMAT CUSREC
*   NUMBER NAME RETRIEVAL TYPE ALTSEQ
*    0001 ARBAL ASCENDING SIGNED NO
*    0002 AREACD DESCENDING ABSVAL NO
       05   CUSREC.
       06   ARBAL          PIC S9(7)V9(2)      COMP-3          CUSREC
       06   AREACD         PIC S9(3)           COMP-3.         CUSREC
       06   BOSTAZ         PIC X(1).                           CUSREC
       06   CNTCT          PIC X(15).                          CUSREC
       06   CRCHKZ         PIC S9(2).                          CUSREC
       06   CSTAT          PIC X(1).                           CUSREC
       06   CUSTNZ         PIC S9(6).                          CUSREC
       06   DLORD          PIC S9(6).                          CUSREC
       06   DSCPCZ         PIC S9(2)V9(3)      COMP-3.         CUSREC
       06   INDUS          PIC S9(2).                          CUSREC
       06   NAME1          PIC X(25).                          CUSREC
       06   NAME2          PIC X(25).                          CUSREC
       06   NAME3          PIC X(25).                          CUSREC
       06   NAME4          PIC X(25).                          CUSREC
       06   PHONE          PIC S9(7)           COMP-3.         CUSREC
       06   PRICIZ         PIC S9(2).                          CUSREC
       06   SHPINZ         PIC X(25).                          CUSREC
       06   SLSMAZ         PIC X(3).                           CUSREC
       06   TAXCDZ         PIC S9(2).                          CUSREC
       06   TERMSZ         PIC S9(2).                          CUSREC
```

Redefinition of Formats

The user should pay particular attention to the REDEFINES clause that may be generated for the ALL-FORMATS or -I-O phrases. Since all formats are redefined on the same area (generally a buffer area), several field names can describe the same area of storage, and unpredictable results can occur if the entire format area is not reinitialized prior to each output operation.

Data items that are subordinate to the data item specified in a MOVE CORRESPONDING statement do not correspond and are not moved when they contain a REDEFINES clause or are subordinate to a redefining item.

To avoid reinitialization, multiple Format 2 COPY statements (DDS or DD) using -I and -O suffixes can be used to create separate areas of storage in the Working-Storage section for each format or format type (input or output). READ INTO and WRITE FROM statements can be used with these record formats. For example:

```
FD ORDER-ENTRY-SCREEN . . .
01 ORDER-ENTRY-RECORD . . .
   .
   .
   .
WORKING-STORAGE SECTION.
01 ORDSFL-I-FORMAT.
   COPY DDS-ORDSFL-I OF DOESCR.
01 ORDSFL-O-FORMAT.
   COPY DDS-ORDSFL-O OF DOESCR.
   .
   .
   .
PROCEDURE DIVISION.
   .
   .
   .
READ SUBFILE ORDER-ENTRY-SCREEN NEXT MODIFIED RECORD
   INTO ORDSFL-I-FORMAT FORMAT IS "ORDSFL"
   AT END SET NO-MODIFIED-SUBFILE-RCD TO TRUE.
   .
   .
   .
MOVE CORR ORDSFL-I TO ORDSFL-O.
REWRITE SUBFILE ORDER-ENTRY-RECORD FROM ORDSFL-O-FORMAT
                           FORMAT IS "ORDSFL" . . .
   .
   .
   .
```

Additional Notes on Field and Format Names

If the generated field-name is a COBOL reserved word, the suffix -DDS is appended to the field-name.

The format-name can be a COBOL reserved word only if the REPLACING phrase is used to change the copied occurrence of the format-name.

The REPLACING phrase cannot be used to change the name of a key field when EXTERNALLY-DESCRIBED-KEY is used.

Floating Point Fields

COBOL treats floating point fields as FILLER. The fields can contain floating point values set outside of COBOL, and a COMP-4 definition is generated to maintain proper alignment in the record, but the data is *not* in binary format. No attempt must be made to use floating point data for processing in the COBOL program.

Floating point key fields are not allowed. In cases where some formats exist with a floating point key field and other formats do not, you should use one or more Format 2 COPY statements with specific format names, rather than using the ALL-FORMATS option.

Note: If you have not specified your own program collating sequence, you may create a record containing floating point fields in your COBOL program by moving LOW-VALUES to the entire record before moving in the values of the non-floating-point fields. This will give the floating point fields in the record a value of zero. Note that the above method is only recommended if valid floating point fields with a value of zero are desirable for your particular application.

Considerations when Using REPLACING with Format 2 COPY Statement

The REPLACING phrase can be used to replace any of the generated COBOL source, including the level numbers and the format-name. (See “REPLACING Phrase” on page 32 for additional information on the REPLACING phrase). However, you should note the following exception:

- When RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified, the REPLACING phrase cannot change the name of a field that is a key.

For example:

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE


   1  000100 01  CUSTOMER-RECORD.                                                          05/05/94
      000200*                                                                            05/05/94
      000300* COPY DDS  W I T H O U T  REPLACING OPTION                                   05/09/94
      000400*                                                                            05/05/94
   2  000500     COPY DDS-CUSMST OF TESTLIB-CUSMSTP.                                     05/05/94
     +000001*    I-O FORMAT:CUSMST     FROM FILE CUSMSTP    OF LIBRARY TESTLIB           CUSMST
     +000002*                          ORDER HEADER RECORD                               CUSMST
   3 +000003       05  CUSMST.                                                          CUSMST
   4 +000004           06 CUST             PIC X(5).                                    CUSMST
     +000005*                          CUSTOMER NUMBER                                   CUSMST
   5 +000006           06 NAME             PIC X(25).                                   CUSMST
     +000007*                          CUSTOMER NAME                                     CUSMST
   6 +000008           06 ADDR             PIC X(20).                                   CUSMST
     +000009*                          CUSTOMER ADDRESS                                  CUSMST
   7 +000010           06 CITY             PIC X(20).                                   CUSMST
     +000011*                          CUSTOMER CITY                                     CUSMST
   8 +000012           06 STATE            PIC X(2).                                    CUSMST
     +000013*                          STATE                                             CUSMST
   9 +000014           06 ZIP              PIC S9(5)        COMP-3.                     CUSMST
     +000015*                          ZIP CODE                                          CUSMST
```

Figure 53. COPY DDS Without the REPLACING Option

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE


      000900* COPY DDS  W I T H  REPLACING OPTION                                          05/05/94
      001000*                                                                            05/05/94
   20 001100     COPY DDS-CUSMST OF TESTLIB-CUSMSTP                                      05/05/94
   21 001200          REPLACING NAME BY ADDR-LINE-1                                      05/05/94
   22 001300                    ADDR BY ADDR-LINE-2                                      05/05/94
   23 001400                    CITY BY ADDR-LINE-3.                                     05/05/94
      +000001*    I-O FORMAT:CUSMST     FROM FILE CUSMSTP    OF LIBRARY TESTLIB          CUSMST
      +000002*                          ORDER HEADER RECORD                              CUSMST
   24 +000003       05  CUSMST.                                                         CUSMST
   25 +000004           06 CUST              PIC X(5).                                  CUSMST
      +000005*                          CUSTOMER NUMBER                                  CUSMST
   26 +000006           06 ADDR-LINE-1             PIC X(25).                          CUSMST
      +000007*                          CUSTOMER NAME                                    CUSMST
   27 +000008           06 ADDR-LINE-2             PIC X(20).                          CUSMST
      +000009*                          CUSTOMER ADDRESS                                 CUSMST
   28 +000010           06 ADDR-LINE-3             PIC X(20).                          CUSMST
      +000011*                          CUSTOMER CITY                                    CUSMST
   29 +000012           06 STATE             PIC X(2).                                  CUSMST
      +000013*                          STATE                                            CUSMST
   30 +000014           06 ZIP               PIC S9(5)        COMP-3.                   CUSMST
      +000015*                          ZIP CODE                                         CUSMST
```

Figure 54. COPY DDS With the REPLACING Option

DDS field names can contain characters that are not allowed in the COBOL language. For example, the field name CUSTN# can be used as an abbreviation for customer number. This format can be used in a Format 2 COPY statement if the REPLACING phrase is used to change the invalid characters to valid COBOL characters.

```
COPY DDS-ALL-FORMATS OF CUSTLIB-PHEADR2
REPLACING CUSTN# BY CUSTNO.
I-O FORMAT:HEDR2      FROM FILE PHEADR2    OF LIBRARY CUSLIB
*    THE KEY DEFINITIONS FOR RECORD FORMAT HEADR2
*    NUMBER NAME RETRIEVAL TYPE ALTSEQ
*    0001 CUSTNO ASCENDING N NO
*    0002 RECCOD DESCENDING ZONE YES
       05 PHEADR2-RECORD PIC X(113).
       05 HEADR2    REDEFINES PHEADR2-RECORD.
          06   CUSTNO  PIC S9(6).
          06   ORDERN  PIC S9(6).
          06   RECCOD  PIC S9(1).
          06   SHIP1   PIC X(25).
          06   SHIP2   PIC X(25).
          06   SHIP3   PIC X(25).
```

End of IBM Extension

Access Path

The description of an externally described file contains the access path that describes how records are to be retrieved from the file. Records can be retrieved based on an arrival sequence (nonkeyed) access path or on a keyed sequence access path.

The arrival sequence access path is based on the order in which the records are stored in the file. Records are added only to the end of the file.

For the keyed sequence access path, the sequence in which records are retrieved from the file is based on the contents of the key field(s) defined in the DDS for the file. For example, in the DDS shown in Figure 47 on page 217, CUST is defined as the key field. The keyed sequence access path is updated whenever records are added, deleted, or the contents of a key field change.

Record Keys and Common Keys

For a keyed sequence access path, one or more fields can be defined in the DDS to be used as the key fields for a record format. All record types in a file do not have to have the same key fields. For example, an order header record can have the ORDER field defined as the key field, and the order detail records can have the ORDER and LINE fields defined as the key fields.

The key for a file is determined by the valid keys for the record types in that file. The file's key is determined in the following manner:

- If all record types in a file have the same number of key fields defined in DDS that are identical in attributes, the *key for the file* consists of all fields in the key for the record types. (The corresponding fields do not have to have the same name.) For example, if the file has three record types and the key for each record type consists of fields A, B, and C, then the file's key consists of fields A, B, and C. That is, the file's key is the same as the records' key.
- If all record types in the file do not have the same key fields, the key for the file consists of the key fields *common* to all record types. For example, a file has three record types and the key fields are defined as follows:
  - REC1 contains key field A.
  - REC2 contains key fields A and B.
  - REC3 contains key fields A, B, and C.

  Then the file's key is field A, the key field common to all record types.
- If no key field is common to all record types, any keyed reference to the file will always return the first record in the file.

In COBOL you must specify a RECORD KEY for an indexed file to identify the record you want to process. COBOL compares the key value with the key of the file or record, and processes the specified operation on the record whose key matches the RECORD KEY value.

When RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified:

- If the FORMAT phrase is specified, the compiler builds the search argument from the key fields in the record area for the specified format
- If the FORMAT phrase is not specified, the compiler builds the search argument from the key fields in the record area for the first record format defined in the program for that file.

Note: For a file containing multiple key fields to be processed in COBOL, the key fields must be contiguous in the record format used by the COBOL program, except when RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified.

Overriding or Adding COBOL Functions to the External Description

In addition to placing the external file description in the program through the use of the Format 2 COPY statement, the user can also use standard record definition and redefinition to describe external files or to provide a group definition for a series of fields. It is the programmer's responsibility to ensure that program described definitions are compatible with the external definitions of the file.

Level Checking

When a COBOL program uses an externally described file, OS/400 provides a level check function. This function ensures that the format has not changed since compilation time.

COBOL always provides the information required by level checking when an externally described file is used (i.e. when a record description was defined for the file by using the Format 2 COPY statement. Only those formats that were copied by the Format 2 COPY statement under the FD for a file are level checked. The level check function will be initiated at run time based on the selection made on the create, change, or override file commands. The default on the create file command is to request level checking. If level checking was requested, level checking occurs on a record format basis when the file is opened. If a level check error occurs, COBOL sets a file status of 90 at OPEN time.

If a file is recreated using an existing format, any existing COBOL programs that use that format can still be used (assuming that no other conflicts such as a change of keys exist) without recompilation.

Note: COBOL does not provide level checking for program described files.

Program Described Files

Records and fields for a program described file are described by coding record descriptions in the File Section of the COBOL program instead of using the Format 2 COPY statement.

The file must be created on the system before the program can be run. This can be done by using one of the Create File commands.

DDS can be used with the Create File commands. For a COBOL indexed file, a keyed access path must be created. This can be done by specifying a key in DDS when the file is created. The record key in COBOL must match the key defined when the file was created.

Specific COBOL File Processing

Printer File Considerations

You can obtain printed output from a COBOL program by issuing WRITE statements to one or more printer files. Each printer file must have a unique name and be assigned to a device of PRINTER or FORMATFILE in the ASSIGN clause of that file's FILE-CONTROL entry. A device of PRINTER must be used for program described files, and a device of FORMATFILE must be used for externally described printer files. The Create Print File (CRTPRTF) command can be used to create a printer file, or one of the IBM-supplied printer device files, such as QPRINT can be used.

The file operations that are valid for a printer file are WRITE, OPEN, and CLOSE. For a complete description of these operations, see Chapter 10, "Procedure Division."

FORMATFILE must be used when the file is an externally described printer file. See "FORMATFILE Files" on page 231 for information on the DDS for externally described printer files.

SPECIAL-NAMES Paragraph and the ADVANCING Phrase

When the mnemonic-name associated with the function-name CSP is specified in the ADVANCING phrase of a WRITE statement for a printer file, it has the same effect as specifying ADVANCING 0 LINES.

When the mnemonic-name associated with the function-name C01 is specified in the ADVANCING phrase of a WRITE statement for a printer file, it has the same effect as specifying ADVANCING PAGE.

The ADVANCING phrase cannot be specified in WRITE statements for files assigned to FORMATFILE.

LINAGE Clause

When the LINAGE clause is specified for a file assigned to PRINTER, all spacing and paging controls are handled internally by compiler generated code. At OPEN time, the printer is positioned to a new physical page and the LINAGE-COUNTER is set to 1. All spacing or paging for following WRITE statements for the file is controlled internally, and the physical page size is ignored. For a file that has a LINAGE clause and is assigned to PRINTER, paging consists of spacing to the end of the logical page (page body) and then spacing past the bottom and top margins.

Use of the LINAGE clause degrades performance. The LINAGE clause should be used only as necessary. If the physical paging is acceptable, the LINAGE clause is not necessary.

The LINAGE clause should not be used for files assigned to FORMATFILE.

FORMATFILE Files

Externally described printer files must be assigned to a device of FORMATFILE. The term FORMATFILE is used because the FORMAT phrase is valid in WRITE statements for the file, and the data formatting is specified in the DDS for the file.

When you have specified a device of FORMATFILE, you can obtain formatting of printed output in two ways:

1. Choose which formats to print in which order by using appropriate values in the FORMAT phrases specified for WRITE statements. For example, use one format once per page to produce a heading, and use another format to produce the detail lines on the page.
2. Choose the appropriate options to be taken when each format is printed by setting indicator values and passing these indicators through the INDICATOR phrase for the WRITE statement. For example, fields may be underlined, blank lines may be produced before or after the format is printed, or the printing of certain fields may be skipped.

The use of external descriptions for printer files has the following advantages over program descriptions:

- Multiple lines can be printed by one WRITE statement. When multiple lines are written by one WRITE statement and the END-OF-PAGE condition is reached, the END-OF-PAGE imperative statement is processed after all of the lines are printed. It is possible to print lines in the overflow area, and onto the next page before the END-OF-PAGE imperative statement is processed.

Figure 55 shows an example of an occurrence of the END-OF-PAGE condition through FORMATFILE.

- Optional printing of fields based on indicator values is possible.
- Editing of field values is easily defined.
- Maintenance of print formats, especially those used by multiple programs, is easier.

Use of the ADVANCING phrase for FORMATFILE files causes a compilation error to be issued. Advancing of lines is controlled in a FORMATFILE file through DDS keywords such as SKIPA and SKIPB, and through the use of line numbers.

For FORMATFILE files, the LINAGE clause is invalid.

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.     FORMATFILE.
    3  000300   AUTHOR.       PROGRAMMER NAME.
    4  000400   INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
    5  000500   DATE-WRITTEN. 08/31/88.
    6  000600   DATE-COMPILED.  08/31/88 16:32:14   .
    7  000700 ENVIRONMENT DIVISION.
    8  000800 CONFIGURATION SECTION.
    9  000900 SOURCE-COMPUTER. IBM-S38.
   10  001000 OBJECT-COMPUTER. IBM-S38.
   11  001100 INPUT-OUTPUT SECTION.
   12  001200 FILE-CONTROL.
   13  001300     SELECT PERSREPT ASSIGN TO FORMATFILE-PERSREPT-SI 1
   14  001400         ORGANIZATION IS SEQUENTIAL.
   15  001500     SELECT PERSFILE ASSIGN TO DATABASE-PERSFILE
   16  001600         ORGANIZATION IS INDEXED
   17  001700         ACCESS MODE IS SEQUENTIAL
   18  001800         RECORD IS EXTERNALLY-DESCRIBED-KEY.
   19  001900 DATA DIVISION.
   20  002000 FILE SECTION.
   21  002100 FD  PERSREPT
   22  002200     LABEL RECORDS ARE STANDARD.
   23  002300 01  PERSREPT-REC.
   24  002400     COPY DDS-ALL-FORMATS-O OF PERSREPT. 2
   25 +000001       05  PERSREPT-RECORD PIC X(130).                                  <-ALL-FMTS
      +000002* OUTPUT FORMAT:HEADING    FROM FILE PERSREPT   OF LIBRARY COB38EX     <-ALL-FMTS
      +000003*                                                                      <-ALL-FMTS
   26 +000004       05  HEADING-O     REDEFINES PERSREPT-RECORD.                    <-ALL-FMTS
   27 +000005           06 ORDERTYPE         PIC X(15).                            <-ALL-FMTS
      +000006* OUTPUT FORMAT:DETAIL     FROM FILE PERSREPT   OF LIBRARY COB38EX     <-ALL-FMTS
      +000007*                                                                      <-ALL-FMTS
   28 +000008       05  DETAIL-O      REDEFINES PERSREPT-RECORD. 3                  <-ALL-FMTS
   29 +000009           06 NAME              PIC X(30).                            <-ALL-FMTS
   30 +000010           06 EMPLNO            PIC S9(6).                            <-ALL-FMTS
   31 +000011           06 BIRTHDATE         PIC X(6).                             <-ALL-FMTS
   32 +000012           06 ADDRESS1          PIC X(35).                            <-ALL-FMTS
   33 +000013           06 MARSTAT           PIC X(1).                             <-ALL-FMTS
   34 +000014           06 SPOUSENAME        PIC X(30).                            <-ALL-FMTS
   35 +000015           06 ADDRESS2          PIC X(20).                            <-ALL-FMTS
   36 +000016           06 NUMCHILD          PIC S9(2).                            <-ALL-FMTS
   37  002500 FD  PERSFILE
   38  002600     LABEL RECORDS ARE STANDARD.
   39  002700 01  PERSFILE-REC.
   40  002800     COPY DDS-ALL-FORMATS-O OF PERSFILE.
   41 +000001       05  PERSFILE-RECORD PIC X(130).                                  <-ALL-FMTS
      +000002*    I-O FORMAT:PERSREC    FROM FILE PERSFILE   OF LIBRARY COB38EX     <-ALL-FMTS
      +000003*                                                                      <-ALL-FMTS
      +000004*THE KEY DEFINITIONS FOR RECORD FORMAT  PERSREC                       <-ALL-FMTS
      +000005*  NUMBER              NAME              RETRIEVAL     TYPE    ALTSEQ  <-ALL-FMTS
      +000006*   0001   EMPLNO                         ASCENDING   SIGNED     NO    <-ALL-FMTS
   42 +000007       05  PERSREC       REDEFINES PERSFILE-RECORD.                    <-ALL-FMTS
   43 +000008           06 EMPLNO            PIC S9(6).                            <-ALL-FMTS
   44 +000009           06 NAME              PIC X(30).                            <-ALL-FMTS
   45 +000010           06 ADDRESS1          PIC X(35).                            <-ALL-FMTS
   46 +000011           06 ADDRESS2          PIC X(20).                            <-ALL-FMTS
   47 +000012           06 BIRTHDATE         PIC X(6).                             <-ALL-FMTS
   48 +000013           06 MARSTAT           PIC X(1).                             <-ALL-FMTS
   49 +000014           06 SPOUSENAME        PIC X(30).                            <-ALL-FMTS
   50 +000015           06 NUMCHILD          PIC S9(2).                            <-ALL-FMTS
```

Figure 55 (Part 1 of 2). Example of the END-OF-PAGE Condition

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   51  002900 WORKING-STORAGE SECTION.
   52  003000 77  HEAD-ORDER                   PIC X(15)
   53  003100                                   VALUE "EMPLOYEE NUMBER".
   54  003200 01  PERSREPT-INDICS.
   55  003300     COPY DDS-ALL-FORMATS-O-INDIC OF PERSREPT.  4
   56 +000001       05  PERSREPT-RECORD.                                          <-ALL-FMTS
      +000002* OUTPUT FORMAT:HEADING    FROM FILE PERSREPT   OF LIBRARY COB38EX   <-ALL-FMTS
      +000003*                                                                   <-ALL-FMTS
      +000004*          06 HEADING-O-INDIC.                                      <-ALL-FMTS
      +000005* OUTPUT FORMAT:DETAIL     FROM FILE PERSREPT   OF LIBRARY COB38EX   <-ALL-FMTS
      +000006*                                                                   <-ALL-FMTS
   57 +000007           06 DETAIL-O-INDIC.                                       <-ALL-FMTS
   58 +000008                07 IN01         PIC 1  INDIC 01.                   <-ALL-FMTS
   59  003400
   60  003500 77  EOF-FLAG                      PIC X(1)
   61  003600                                   VALUE "0".
   62  003700    88  NOT-END-OF-FILE            VALUE "0".
   63  003800    88  END-OF-FILE                VALUE "1".
   64  003900 77  MARRIED                      PIC X(1)
   65  004000                                   VALUE "M".
   66  004100
   67  004200 PROCEDURE DIVISION.
       004300 FIRST-SECT SECTION.
       004400 FIRST-PARA.
   68  004500     OPEN INPUT PERSFILE
       004600          OUTPUT PERSREPT.
   69  004700     PERFORM HEADING-LINE.
   70  004800     PERFORM PROCESS-RECORD UNTIL END-OF-FILE.
   71  004900     CLOSE  PERSFILE
       005000            PERSREPT.
   72  005100     STOP RUN.
       005200
       005300 PROCESS-RECORD.
   73  005400     READ PERSFILE AT END SET END-OF-FILE TO TRUE.
   75  005500     IF NOT-END-OF-FILE THEN
   76  005600         PERFORM PRINT-RECORD. _____
       005700
       005800 PRINT-RECORD.
   77  005900     MOVE CORR PERSREC TO DETAIL-O.  6
   78  006000     IF MARSTAT IN PERSFILE-REC IS EQUAL MARRIED THEN  7
   79  006100         MOVE B"1" TO IN01 IN DETAIL-O-INDIC
       006200     ELSE
   80  006300         MOVE B"0" TO IN01 IN DETAIL-O-INDIC.  8
   81  006400     WRITE PERSREPT-REC FORMAT IS "DETAIL"
       006500         INDICATORS ARE DETAIL-O-INDIC
   82  006600           AT EOP PERFORM HEADING-LINE. _________
       006700 HEADING-LINE.
   83  006800         MOVE HEAD-ORDER TO ORDERTYPE
   84  006900         WRITE PERSREPT-REC FORMAT IS "HEADING".
       007000
                         * * * * *  E N D  O F  S O U R C E  * * * * *
```

1 The externally described printer file is assigned to device FORMATFILE.

2 The Format 2 COPY statement, is used to copy the fields for the printer file into the program.

3 Note that although the fields in format DETAIL will be printed on 3 separate lines, they are defined in one record.

4 COPY-DDS is used to copy the indicators used in the printer file into the program.

5 Paragraph PROCESS-RECORD processes PRINT-RECORD for each employee record.

6 All fields in the employee record are moved to the record for format DETAIL.

7 If the employee is married, indicator 01 is turned on; otherwise the indicator is turned off, preventing the spouse's name field in DETAIL from being printed.

8 Format DETAIL is printed with indicator 01 passed to control printing.

9 If the number of lines per page has been exceeded, END-OF-PAGE occurs. The format HEADING is printed on a new page.

Figure 55 (Part 2 of 2). Example of the END-OF-PAGE Condition

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7784-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Not (N)	Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (b/I/N/K/R/O)	Reserved	Name	Reference (R)	Length	Data Type (b/A/P/S/B A/B/X/Y/N/I/U/W)	Decimal Positions	Usage (b/O/I/B/H/M)	Location: Line	Pos	Functions
	A	*		PHYSICAL					F		FILE DDS	FOR	PERSONNEL						FILE IN FORMATFILE EXAMPLE
	A																		
	A	*							R		PERSREC								
	A										EMPLNO		6	S					
	A										NAME		30						
	A										ADDRESS1		35						
	A										ADDRESS2		20						
	A										BIRTHDATE		6						
	A										MARSTAT		1						
	A										SPOUSENAME		30						
	A										NUMCHILD		2	S					
	A								K		EMPLNO								
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		

Figure 56 (Part 1 of 2). Example of the Use of Externally Described Printer Files Assigned to a Device of FORMATFILE

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7764-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Name Type (b/H/F/E/O)	Reserved	Name	Reference (R)	Length	Data Type (b A/P/S/B A/S/X/Y/N/I/W)	Decimal Positions	Usage (b/I/O/I/B/H/W)	Location: Line	Location: Pos	Functions
	A	*		PRINTER					F		FILE DDS FOR		FORMATFILE EXAMPLE						
	A	*																	
	A																	1	INDARA REF(PERSFILE)
	A								R		HEADING 2								SKIPB(1) SPACEA(3) 3
	A																	15	'PERSONNEL LISTING'
	A																		UNDERLINE
	A																	33	'- ORDERED BY'
	A										ORDERTYPE		15					46	
	A																	80	DATE EDTCDE(Y)
	A																	93	TIME 4
	A																	115	'PAGE:'
	A																	+1	PAGNBR EDTCDE(3)
	A	*																	
	A								R		DETAIL 5								SPACEA(3) 6
	A	*			LINE	1													
	A																	1	'NAME:'
	A										NAME	R						11	UNDERLINE
	A																	55	'EMPLOYEE NUMBER:'
	A										EMPLNO	R						73	
	A																	87	'DATE OF BIRTH:'
	A										BIRTHDATE	R						103	SPACEA(1) 7
	A	*			LINE	2													
	A																	1	'ADDRESS:'
	A										ADDRESS1	R						11	
	A																	55	'MARITAL STATUS:'
	A										MARSTAT	R						73	
	A		01															87	'SPOUSE''S NAME:'
	A		01	8							SPOUSENAME	R						103	
	A	*			LINE	3													
	A										ADDRESS2	R						11	SPACEB(1)
	A																	55	'CHILDREN:'
	A										NUMCHILD	R						73	EDTCDE(3) 9
	A																		
	A																		
	A																		

1 INDARA specifies that a separate indicator area is to be used for the file.

2 HEADING is the format name which provides headings for each page.

3 SKIPB(1) and SPACEA(3) are used to:

1. Skip to line 1 of the next page before format HEADING is printed.
2. Leave 3 blank lines after format HEADING is printed.

4 DATE, TIME and PAGNBR are used to have the current date, time and page number printed automatically when format HEADING is printed.

5 DETAIL is the format name used to print the detail line for each employee in the personnel file.

6 SPACEA(3) causes 3 lines to be left blank after each employee detail line.

7 SPACEA(1) causes a blank line to be printed after the field BIRTHDATE is printed. As a result, subsequent fields in the same format are printed on a new line.

8 01 means that these fields are printed only if the COBOL program turns indicator 01 on and passes it when format DETAIL is printed.

9 EDTCDE(3) is used to remove leading zeros when printing this numeric field.

Figure 56 (Part 2 of 2). Example of the Use of Externally Described Printer Files Assigned to a Device of FORMATFILE

DISK and DATABASE File Considerations

Data base files, which are associated with the COBOL devices of DATABASE and DISK, can be:

- Externally described files, whose fields are described to OS/400 through DDS
- Program described files, whose fields are described in the program that uses the file.

All data base files are created by OS/400 Create File commands.

DATABASE versus DISK Files

Assigning a file to DISK in COBOL restricts the user to traditional DISK processing. The use of DATABASE as the device permits the user to make use of the special System/38 environment COBOL data base features such as formats and duplicate record keys.

Processing Methods for DISK and DATABASE Files

COBOL Indexed Files

An indexed file is a file whose access path is built on key values. The user must create a keyed access path for an indexed file by using DDS.

To write standard ANS COBOL X3.23-1974 to access an indexed file, the file must be created with certain characteristics. The following table lists these characteristics and what controls them.

Characteristic	Control
The file must be a physical file.	The CL command CRTPF
The file cannot have records with duplicate key values.	The DDS keyword UNIQUE
The file cannot be a shared file.	The CL command CRTPF
A key must be defined for the file.	DDS
Keys must be in ascending sequence.	DDS
Keys must be contiguous within the record.	DDS
Key fields must be alphanumeric. They cannot be numeric only.	DDS
The value of the key used for sequencing must include all 8 bits of every byte.	DDS
A starting position for retrieving records cannot be specified.	The CL command OVRDBF
Select/omit level keywords cannot be used for the file.	DDS

An indexed file is identified by the ORGANIZATION IS INDEXED clause of the SELECT statement.

The key fields identify the records in an indexed file. The user specifies the key field in the RECORD KEY clause of the SELECT statement. The RECORD KEY data item must be defined within a record description for the indexed file. If there are multiple record descriptions for the file, only one need contain the RECORD KEY data-name.

However, the same positions within the record description that contain the RECORD KEY data item are accessed in the other record descriptions as the KEY value for any references to the other record descriptions for that file.

An indexed file can be accessed sequentially, randomly by key, or dynamically.

Valid RECORD KEYS: The DDS for the file specifies the field(s) to be used as the key field. If the file has multiple key fields, the key fields must be contiguous in each record unless RECORD KEY IS EXTERNALLY-DESCRIBED-KEY is specified.

When the DDS specifies only one key field for the file, the RECORD KEY must be a single field of the same length as the key field defined in the DDS.

If a Format 2 COPY statement is specified for the file, the RECORD KEY clause must specify one of the following:

- The name used in the DDS for the key field if the name is not a COBOL reserved word.
- The name used in the DDS for the key field with -DDS added to the end if the name is a COBOL reserved word.
- The data-name defined with the proper length and at the proper location in a program described record description for the file.
- EXTERNALLY-DESCRIBED-KEY. This keyword specifies that the key(s) defined in DDS for each record format are to be used for accessing the file. These keys can be noncontiguous. They can be defined at different positions within the record format.

When the DDS specifies multiple contiguous key fields, the RECORD KEY data-name must be a single field with its length equal to the sum of the lengths of the multiple key fields in the DDS. If a Format 2 COPY statement is specified for the file, there must also be a program described record description for the file that defines the RECORD KEY data-name with the proper length and at the proper position in the record.

Referring to a Partial Key

A generic START statement allows the use of a partial key. The KEY IS phrase is required.

"START Statement" in Chapter 10, "Procedure Division" lists the rules for specifying a search argument that refers to a partial key.

Figure 57 on page 238 shows an example of generic STARTs using a program described file.

Figure 58 on page 238 shows an example of generic STARTs using an externally described file.

```
5763CB1                          COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    7  000700 FILE-CONTROL.
    8  000800     SELECT FILE-1 ASSIGN TO DISK-FILE1
    9  000900     ACCESS IS DYNAMIC RECORD KEY IS FULL-NAME IN FILE-1
   10  001000     ORGANIZATION IS INDEXED.
   11  001100 DATA DIVISION.
   12  001200 FILE SECTION.
   13  001300 FD  FILE-1 LABEL RECORDS ARE STANDARD.
   14  001400 01  RECORD-DESCRIPTION.
   15  001500     03 FULL-NAME.
   16  001600        05 LAST-AND-FIRST-NAMES.
   17  001700           07 LAST-NAME             PIC X(20).
   18  001800           07 FIRST-NAME            PIC X(20).
   19  001900        05 MIDDLE-NAME              PIC X(20).
   20  002000     03 LAST-FIRST-MIDDLE-INITIAL-NAME REDEFINES FULL-NAME
   21  002100                                    PIC X(41).
   22  002200     03 REST-OF-RECORD
       002300/
   23  002400 PROCEDURE DIVISION.
       002500 START-PROGRAM.
   24  002600     OPEN INPUT FILE-1.
       002700*
       002800* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
       002900* "SMITH"
   25  003000     MOVE "SMITH" TO LAST-NAME.
   26  003100     START FILE-1 KEY IS EQUAL TO LAST-NAME
   27  003200           INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR " LAST-NAME
   28  003300                       GO-TO ERROR ROUTINE.
       003400*
       003500*
       003600*
       003700*
       003800* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
       003900* "SMITH" AND A FIRST NAME OF "ROBERT"
   29  004000     MOVE "SMITH" TO LAST-NAME.
   30  004100     MOVE "ROBERT" TO FIRST-NAME.
   31  004200     START FILE-1 KEY IS EQUAL TO LAST-AND-FIRST-NAMES
   32  004300           INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
       004400                               LAST-AND-FIRST-NAMES
   33  004500                       GO-TO ERROR ROUTINE.
       004600*
       004700*
       004800*
       004900*
       005000* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
       005100* "SMITH", AND A FIRST NAME OF "ROBERT", AND A MIDDLE INITIAL OF "M"
   34  005200     MOVE "SMITH" TO LAST-NAME.
   35  005300     MOVE "ROBERT" TO FIRST-NAME.
   36  005400     MOVE "M" TO MIDDLE-NAME.
   37  005500     START FILE-1 KEY IS EQUAL TO LAST-AND-FIRST-MIDDLE-INITIAL-NAME
   38  005600           INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
       005700                               LAST-FIRST-MIDDLE-INITIAL-NAME
   39  005800                       GO-TO ERROR ROUTINE.
       005900
       006000
       006100 ERROR-ROUTINE.
   40  006200     STOP-RUN.
```

Figure 57. Generic STARTs Using a Program Described File

```
                                DATA DESCRIPTION SOURCE
     SEQNBR *... ... 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7 ... ... 8  DATE
        100      A                                      UNIQUE
        200      A          R RDE                       TEXT('RECORD DESCRIPTION')
        300      A            FNAME         20         TEXT('FIRST NAME')
        400      A            MINAME         1         TEXT('MIDDLE INITIAL NAME')
        500      A            MNAME         19         TEXT('REST OF MIDDLE NAME')
        600      A            LNAME         20         TEXT('LAST NAME')
        700      A            PHONE         10  0      TEXT('PHONE NUMBER')
        800      A            DATA          40         TEXT('REST OF DATA')
        900      A          K LNAME
       1000      A          K FNAME
       1100      A          K MINAME
       1200      A          K MNAME
```

Figure 58 (Part 1 of 2). Generic STARTs Using an Externally Described File

```
STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   7  000700 FILE-CONTROL.
   8  000800     SELECT FILE-1 ASSIGN TO DATABASE-NAMES
   9  000900     ACCESS IS DYNAMIC RECORD KEY IS EXTERNALLY-DESCRIBED-KEY
  10  001000     ORGANIZATION IS INDEXED.
  11  001100 DATA DIVISION.
  12  001200 FILE SECTION.
  13  001300 FD  FILE-1 LABEL RECORDS ARE STANDARD.
  14  001400 01  RECORD-DESCRIPTION
  15  001500     COPY DDS-RDE IN NAMES-PUBS.
  17 +000001                                                                      RDE
     +000002*    FROM FILE NAMES      OF LIBRARY PUBS                            RDE
     +000003*    RECORD DESCRIPTION                                              RDE
  18 +000004       05  RDE.                                                      RDE
     +000005*    RECORD KEY FOR INDEXED FILE, KEY'0002 KEY FIELD NAME FNAME     .  RDE
  19 +000006           06 FNAME             PIC X(20).                          RDE
     +000007*    FIRST NAME                                                      RDE
     +000008*    RECORD KEY FOR INDEXED FILE, KEY'0003 KEY FIELD NAME MINAME    .  RDE
  20 +000009           06 MINAME            PIC X(1).                           RDE
     +000010*    MIDDLE INITIAL NAME                                             RDE
     +000011*    RECORD KEY FOR INDEXED FILE, KEY'0004 KEY FIELD NAME MNAME     .  RDE
  21 +000012           06 MNAME             PIC X(19).                          RDE
     +000013*    REST OF MIDDLE NAME                                             RDE
     +000014*    RECORD KEY FOR INDEXED FILE, KEY'0001 KEY FIELD NAME LNAME     .  RDE
  22 +000015           06 LNAME             PIC X(20).                          RDE
     +000016*    LAST NAME                                                       RDE
  23 +000017           06 PHONE             PIC S9(10).      COMP-3             RDE
     +000018*    PHONE NUMBER                                                    RDE
  24 +000019           06 DATA-DDS          PIC X(40).                          RDE
     +000020*    REST OF DATA                                                    RDE
  25  001600 66  MIDDLE-NAME RENAMES MINAME THRU MNAME.
      001700/
  26  001800 PROCEDURE DIVISION.
      001900 START PROGRAM.
  27  002000     OPEN INPUT FILE-1.
      002100*
      002200* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME
      002300* OF "SMITH"
  28  002400     MOVE "SMITH" TO LNAME.
  29  002500     START FILE-1 KEY IS EQUAL TO LNAME
  30  002600           INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR " LNAME
  31  002700                       GO TO ERROR-ROUTINE.
      002800*          .
      002900*          .
      003000*          .
      003100*
      003200* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME
      003300* OF "SMITH" AND A FIRST NAME OF "ROBERT"
  32  003400     MOVE "SMITH" TO LNAME.
  33  003500     MOVE "ROBERT" TO FNAME.
  34  003600     START FILE-1 KEY IS EQUAL TO LNAME, FNAME
  35  003700           INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
      003800                               LNAME " " FNAME
  36  003900                       GO TO ERROR-ROUTINE.
      004000*          .
      004100*          .
      004200*          .
      004300*
      004400* POSITION THE FILE STARTING WITH RECORDS THAT HAVE A LAST NAME OF
      004500* "SMITH", A FIRST NAME OF "ROBERT", AND A MIDDLE INITIAL OF "M"
  32  004600     MOVE "SMITH" TO LNAME.
  33  004700     MOVE "ROBERT" TO FNAME.
  33  004800     MOVE "M" TO MINAME.
  34  004900     START FILE-1 KEY IS EQUAL TO LNAME, FNAME, MINAME
  35  005000           INVALID KEY DISPLAY "NO DATA IN SYSTEM FOR "
      005100                               LNAME SPACE FNAME SPACE MINAME
  42  005200                       GO TO ERROR-ROUTINE.
      005300
      005400
      005500 ERROR-ROUTINE.
      005600     STOP-RUN.
```

Figure 58 (Part 2 of 2). Generic STARTs Using an Externally Described File

Logical File Considerations

When a logical file with multiple record formats, each having associated key fields, is processed as an indexed file in COBOL, the following restrictions and considerations apply:

- The FORMAT phrase must be specified on all WRITE statements to the file.
- If the access mode is RANDOM or DYNAMIC, and the DUPLICATES phrase is not specified for the file, the FORMAT phrase must be specified on all DELETE and REWRITE statements.
- When the FORMAT phrase is not specified, only the portion of the RECORD KEY data item that is common to all record formats for the file is used by the system as the key for the I-O statement. When the FORMAT phrase is specified, only the portion of the RECORD KEY data item that is defined for the specified record format is used by the system as the key.
- When *NONE is specified as the first key field for any format in a file, records can only be accessed sequentially. When a file is read randomly:
  - If a format name is specified, the first record with the specified format is returned.
  - If a format name is not specified, the first record in the file is returned.

  In both cases, the value of the RECORD KEY data item is ignored.
- For a program defined key field:
  - Key fields within each record format must be contiguous.
  - The first key field for each record format must begin at the same relative position within each record.
  - The length of the RECORD KEY data item must be equal to the length of the longest key for any format in the file.
- For an EXTERNALLY-DESCRIBED-KEY:
  - Key fields within each record format can be noncontiguous.
  - The key fields can begin at different positions in each record format.

Figure 59 on page 241 and Figure 60 on page 242 show examples of how to use DDS to describe the access path for indexed files.

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7764-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

A

Sequence Number	Form Type	Conditioning: And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Conditioning: Condition Name: Not (N)	Conditioning: Condition Name: Indicator	Name Type (b/R/K/S/C/O)	Reserved	Name	Reference (R)	Length	Data Type (b/A/P/S/B/A/E/I/X/Y/N/U/W)	Decimal Positions	Usage (b/O/I/B/N/M)	Location: Line	Location: Pos	Functions
	A								R		FORMATA								PFILE(ORDDTLP)
	A																		TEXT('ACCESS PATH FOR INDEXED FILE')
	A										FLDA		14						
	A										ORDERN		5	S	0				
	A										FLDB		101						
	A								K		ORDERN								
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		

Figure 59 (Part 1 of 2). Using Data Description Specifications to Define the Access Path for an Indexed File

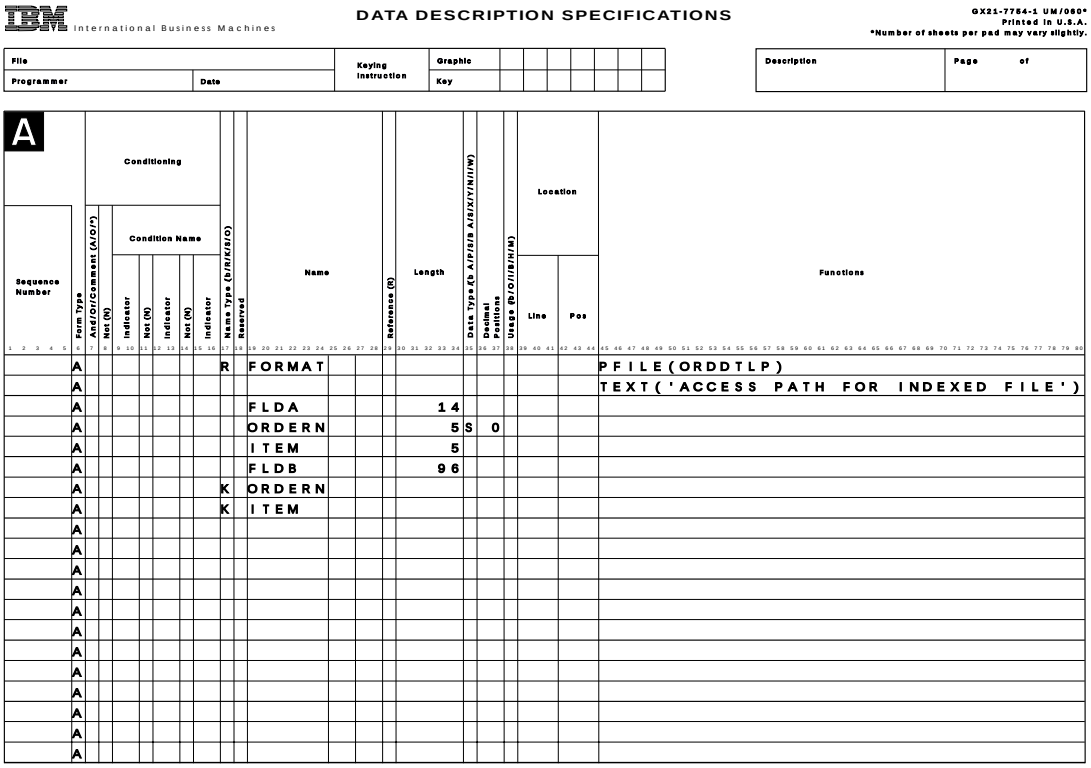
Data description specifications must be used to create the access path for a program described indexed file.

In the DDS for the record format FORMATA for the logical file ORDDTLL, the field ORDERN, which is five digits long, is defined as the key field. The definition of ORDERN as the key field establishes the keyed access for this file. Two other fields, FLDA and FLDB, describe the remaining positions in this record as character fields.

The program described input field ORDDTLL is described in the FILE-CONTROL section in the SELECT clause as an indexed file.

The COBOL descriptions of each field in the FD entry must agree with the corresponding description in the DDS file. The RECORD KEY data item must be defined as a five-digit numeric integer beginning in position 15 of the record.

Figure 59 (Part 2 of 2). Using Data Description Specifications to Define the Access Path for an Indexed File



IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7784-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Condition Name: Not (N)	Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (R/K/S/I/O)	Reserved	Name	Reference (R)	Length	Data Type (B/P/S/B A/E/F/X/Y/N/U/W)	Decimal Positions	Usage (B/O/I/R/H/M)	Location: Line	Location: Pos	Functions
	A								R		FORMAT								PFILE(ORDDTLP)
	A																		TEXT('ACCESS PATH FOR INDEXED FILE')
	A										FLDA		14						
	A										ORDERN		5	S	0				
	A										ITEM		5						
	A										FLDB		96						
	A								K		ORDERN								
	A								K		ITEM								
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		

Figure 60. Using Data Description Specifications to Define the Access Path (a Composite Key) for an Indexed File

In this example, the data description specifications define two key fields for the record format FORMAT in the logical file ORDDTLL. For the two fields to be used as a composite key for a program described indexed file, the key fields must be contiguous in the record.

The COBOL description of each field must agree with the corresponding description in the DDS file. A ten-character item beginning in position 15 of the record must be defined in the RECORD KEY clause of the file-control entry. The COBOL descriptions of the DDS fields ORDERN and ITEM would be subordinate to the 10-character item defined in the RECORD KEY clause.

COBOL Relative File

A COBOL relative file is a file to be processed by a relative record number. To process a file by relative record number, ORGANIZATION IS RELATIVE must be specified in the SELECT statement for the file. A relative file can be accessed sequentially, randomly by record number, or dynamically.

To write standard ANS COBOL X3.23-1974 to access a relative file, the file must be created with certain characteristics. The following table lists these characteristics and what controls them.

Characteristic	Control
The file must be a physical file.	The CL command CRTPF
The file cannot be a shared file.	The CL command CRTPF
No key can be specified for the file.	DDS
A starting position for retrieving records cannot be specified.	The CL command OVRDBF

Characteristic	Control
Select/omit level keywords cannot be used for the file.	DDS

For a COBOL file with an organization of RELATIVE, the Reorganize Physical File Member (RGZPFM) CL command can:

- Remove all deleted records from the file. Since COBOL initializes all relative file records to deleted records, any record that has not been explicitly written will be removed from the file. This causes the relative record numbers of all records after the first deleted record in the file to change.
- Change the relative record numbers if the file has a key and the arrival sequence is changed to match a key sequence (with the KEYFILE parameter).

Either result of the CL command RGZPFM causes the COBOL concept of a relative file to change.

COBOL Sequential File

A COBOL sequential file is a file in which records are processed in the order in which they were placed in the file; that is, in arrival sequence. For example, the tenth record placed in the file occupies the tenth record position and is the tenth record to be processed. To process a file as a sequential file, ORGANIZATION IS SEQUENTIAL must be specified in the SELECT clause, or the ORGANIZATION clause can be omitted. A sequential file can only be accessed sequentially.

To write standard ANS COBOL X3.23-1974 to access a sequential file, the file must be created with certain characteristics. The following table lists these characteristics and what controls them.

Characteristic	Control
The file must be a physical file.	The CL command CRTPF
The file cannot be a shared file.	The CL command CRTPF
The device specified in the assignment-name must match the actual device to which the file is assigned.	
No key can be specified for the file.	DDS
The file must have a file-type of data.	The CL command CRTPF
Field editing cannot be used.	DDS
Line and position information cannot be specified.	DDS
Spacing and shipping keywords cannot be specified.	DDS
Indicators cannot be used.	DDS
System-supplied functions such as date, time, and page number cannot be used.	DDS
Select/omit level keywords cannot be used for the file.	DDS

COBOL File Organization and AS/400 File Access Path Considerations

A file with a keyed sequence access path can be processed in COBOL as a file with INDEXED, RELATIVE, or SEQUENTIAL organization.

To process a keyed sequence file as a relative file in COBOL, the file must be a physical file, or a logical file whose members are based on one physical file member. To process a keyed sequence file as a sequential file in COBOL, the file must be a physical file, or a logical file that is based on one physical file member and that does not contain select/omit logic.

A file with an arrival sequence access path can be processed in COBOL as a file with RELATIVE or SEQUENTIAL organization. However, the file must be a physical file or a logical file where each member of the logical file is based on only one physical file member.

When sequential access is specified for a logical file, records in the file are accessed through the access path created by the user with create file options.

File Processing Methods

Figure 61 on page 245 shows the valid processing methods and expected operation for combinations of organization, access mode, open state, I-O verb, and I-O verb modifiers.

Note: All physical data base files that are opened for output are cleared. Data base files with RELATIVE organization are also initialized with deleted records, which is necessary for successful relative file processing on the AS/400 system. Relative files should be cleared and initialized with deleted records before they are used when the first OPEN statement for the file is not OPEN OUTPUT. The RECORDS parameter in the INZPFM command must specify *DLT. Overrides are applied when COBOL processes the clear and initialize operations, but overrides are not applied when the user processes the clear and initialize operations with CL commands.

Users can expect lengthy delays in OPEN OUTPUT processing for extremely large relative files (over 1,000,000 records), such as when *NOMAX is specified on the create file command.

ORG	ACC	DEV	OPEN	READ	WRITE	START	REWRITE	DELETE	CLOSE	FORMAT	SELECT CLAUSE KEY IS
S	S	ANY	INPUT	X					X		
S	S	ANY	OUTPUT		X(F1)				X	A1	
S	S	ANY	I-O	X			X		X		
S	S	ANY	EXTEND		X				X		
I	S	D/DB	INPUT	X		X			X	B1	C1
I	S	D/DB	OUTPUT		X(F1)				X	B1	C1
I	S	D/DB	I-O	X		X	X	X	X	B1	C1
I	R	D/DB	INPUT	X					X	B1	D1
I	R	D/DB	OUTPUT		X(F1)				X	B1	D1
I	R	D/DB	I-O	X	X		X	X	X	B1	D1
I	D	D/DB	INPUT	X		X			X	B1	D1
I	D	D/DB	OUTPUT		X(F1)				X	B1	D1
I	D	D/DB	I-O	X	X	X	X	X	X	B1	D1
R	S	D/DB	INPUT	X		X			X		C1
R	S	D/DB	OUTPUT		X(G1)				X		C1
R	S	D/DB	I-O	X		X	X	X	X		C1
R	R	D/DB	INPUT	X					X		E1
R	R	D/DB	OUTPUT		X(G1)				X		E1
R	R	D/DB	I-O	X	X		X	X	X		E1
R	D	D/DB	INPUT	X		X			X		E1
R	D	D/DB	OUTPUT		X(G1)				X		E1
R	D	D/DB	I-O	X	X	X	X	X	X		E1
T	S	W	I-O	X	X				X	H1	
T	D	W	I-O	X(K1)	X(K1)		X		X	I1	J1

ORG:	ACC:	DEV:
S = Sequential R = Relative I = Indexed T = TRANSACTION	S = Sequential R = Random D = Dynamic	ANY = Any Device D = DISK DB = DATABASE W = WORKSTATION

Figure 61. Processing Methods Summary Chart

The following paragraphs explain the keys used in Figure 61.

X The combination is allowed.

A1 The FORMAT phrase is required for FORMATFILE files with multiple formats, and is not allowed for all other device files.

B1 The FORMAT phrase is optional for DATABASE files, and not allowed for DISK files. If the FORMAT phrase is not specified, the default format name of the file is used. The default format name of the file is the first format name defined in the file.

The special register, DB-FORMAT-NAME, can be used to retrieve the format name used on the last successful I-O operation.

C1 The SELECT clause KEY phrase is ignored except for the START statement. If the KEY phrase is not specified on the START statement, the RECORD KEY phrase or the RELATIVE KEY phrase in the SELECT clause is used and KEY = is assumed.

D1 The SELECT clause KEY phrase is used except for the START statement. If the KEY phrase is not specified on the START statement, the RECORD KEY phrase in the SELECT clause is used and KEY = is assumed.

NEXT, PRIOR, FIRST, or LAST can be specified only for the READ statement for DATABASE files with DYNAMIC access. If NEXT, PRIOR, FIRST, or LAST is specified, the SELECT clause KEY phrase is ignored.

E1 The SELECT clause RELATIVE KEY phrase is used.

The NEXT phrase can be specified only for the READ statement for a file with DYNAMIC access mode. If NEXT is specified, the SELECT clause KEY phrase is ignored.

The RELATIVE KEY data item is updated with the relative record number for files with sequential access on READ operations.

F1 A physical file opened for output is cleared.

G1 A physical file opened for output is cleared and initialized to deleted records.

H1 The FORMAT phrase is required for the WRITE statement.

I1 The FORMAT phrase is required to distinguish between the subfile records and the subfile control record. The WRITE FORMAT IS control-record-format-name displays the subfile, but a READ FORMAT IS control-record-format-name is required to allow data to be entered and to cause the operator input for the subfile records on the display to be placed in the subfile.

J1 The SELECT clause RELATIVE KEY phrase is used for READ, WRITE, and REWRITE statements that use the SUBFILE phrase, except that the READ SUBFILE NEXT MODIFIED uses the current system relative record number rather than the RELATIVE KEY data item. The RELATIVE KEY data item is updated with the relative record number for subfile records for READ statements with the NEXT MODIFIED clause.

K1 The SUBFILE phrase is required when an I-O operation deals with a particular record rather than an entire file.

Descending File Considerations

Files created with a descending keyed sequence (in DDS) cause the READ statement NEXT, PRIOR, FIRST, and LAST phrases to work in a fashion exactly opposite that of a file with an ascending key sequence. For example, READ FIRST retrieves the record with the highest key value, and READ LAST retrieves the record with the lowest key value. Files with a descending key sequence also cause the START qualifiers to work in the opposite manner. For example, START GREATER THAN positions the current record pointer to a record with a key less than the current key.

Commitment Control Considerations

Commitment control is a function that allows:

- Synchronization of changes to data base files within the same job
- Cancellation of changes that should not be permanently entered into the data-base
- Locking of records being changed until changes are complete
- Techniques for recovering from job or system failure.

In some applications, it is desirable to synchronize changes to data base records. If the program determines the changes are valid, the changes are then permanently made to the data base (a `COMMIT` statement is processed). If the changes are not valid, or if a problem occurs during processing, the changes can be canceled (a `ROLLBACK` statement is processed). (When a file is cleared after being opened for `OUTPUT`, processing of a `ROLLBACK` does not restore cleared records to the file.) Changes made to records in a file that is *not* under commitment control are always permanent. Such changes are never affected by subsequent `COMMIT` or `ROLLBACK` statements.

Each point where a `COMMIT` or `ROLLBACK` is successfully processed is a commitment boundary. (If no `COMMIT` or `ROLLBACK` has yet been issued in a program, a commitment boundary is created by the first open of any file under commitment control.) The committing or rolling back of changes only affects changes made since the previous commitment boundary.

The synchronizing of changes at commitment boundaries makes restart or recovery procedures after a failure easier. For more information see “Recovery after a Failure” on page 257.

When commitment control is used for data base files, records in those files are subject to either a high lock level `LCKLVL (*ALL)` or a low lock level `LCKLVL(*CHG)`. With a low lock level (`*CHG`), all records that are changed (rewritten, deleted, or added) in files under commitment control are locked until a `COMMIT` or `ROLLBACK` statement is successfully processed. With a high lock level (`*ALL`), *all* records accessed, whether for input or output, are locked until a `COMMIT` or `ROLLBACK` is successfully processed. For both record locking levels, no other job can modify data in locked records until the `COMMIT` or `ROLLBACK` has been successfully completed. (A locked record can only be modified within the same job and through the same physical or logical file.)

The lock level also governs whether locked records can be read. With a high lock level (`*ALL`), you cannot read locked records in a data base file.

With a low lock level (`*CHG`), you can read locked records in a data base file, provided the file is opened as `INPUT` in your job.

Other jobs, where files are *not* under commitment control, can always read locked records, regardless of the lock level used, provided the files are opened as `INPUT`. Because it is possible in some cases for other jobs to read locked records, data can be accessed *before it is permanently committed to a data base.* If a `ROLLBACK` statement is processed *after* another job has read locked records, the data accessed will not reflect the contents of the data base.

Figure 62 on page 248 shows record locking considerations for files with and without commitment control.

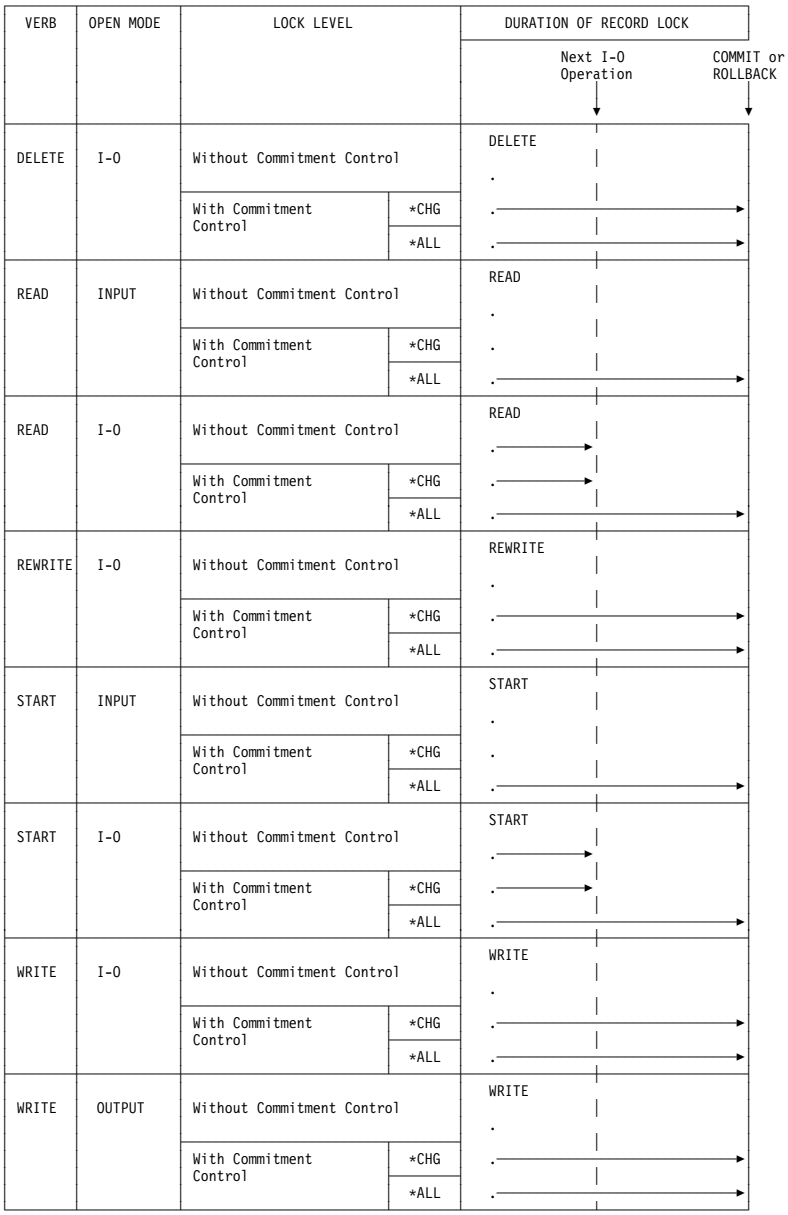


Figure 62. Record Locking Considerations with and without Commitment Control

Note: A `WRITE` is not considered an update operation; therefore, the record lock will not be released.

A file under commitment control can be closed or opened without affecting the status of changes made since the last commitment boundary. A `COMMIT` must still be issued to make the changes permanent, or a `ROLLBACK` issued to cancel the changes. A `COMMIT` statement, when processed, leaves files in the same open or closed state as before processing.

All files under commitment control within the same job must be journaled to the same journal.

Commitment control must also be specified outside the COBOL language through the OS/400 control language (CL). The Begin Commitment Control (BGNCMTCTL) CL command establishes the capability for commitment control and sets the level of record locking at the high level (*ALL), or the low level (*CHG). The BGNCMTCTL command does not automatically initiate commitment control for a file. That file must also be specified in the COMMITMENT CONTROL clause of the I-O-CONTROL paragraph within the COBOL program. The commitment control environment is normally ended by using the End Commitment Control (ENDCMTCTL) CL command. This causes any uncommitted changes for data base files under commitment control to be cancelled. (An implicit ROLLBACK is processed.)

Note: The ability to prevent reading of uncommitted data that has been changed is a function of commitment control and is only available if you are running under commitment control. Normal (non-commit) data base support is not changed by the commitment control extension, and allows reading of locked records when a file that is opened only for input is read. Try to use files consistently. Typically, files should always be run under commitment control or never be run under commitment control.

Figure 63 on page 250 illustrates a possible use of commitment control in a banking environment. The program processes transactions for transferring funds from one account to another. If no problems occur during the transaction, the changes are committed to the data base file. If the transfer is invalid due to improper account number or insufficient funds, a ROLLBACK is issued to cancel the changes.

IBM International Business Machines

DATA DESCRIPTION SPECIFICATIONS

GX21-7784-1 UM/060*
Printed in U.S.A.
*Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic								Description	Page of
Programmer	Date		Key									

Sequence Number	Form Type	And/Or/Comment (A/O/C)	Conditioning: Not (N)	Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (b/I/N/K/S/O)	Reserved	Name	Reference (R)	Length	Data Type (b, A/P/S/B, A/B/X/Y/N/I/1/W)	Decimal Positions	Usage (b/I/O/I/B/H/M)	Location: Line	Pos	Functions
1 2 3 4 5	6	7	8	9 10	11	12 13	14	15 16	17	18	19 20 21 22 23 24 25 26 27 28	29	30 31 32 33 34	35	36 37	38	39 40 41	42 43 44	45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
	A	*		PROMPT							SCREEN FILE		NAME	'ACCTFMTS'					
	A	*																	
	A																	1	INDARA
	A								R		ACCTPMT								
	A																		TEXT('CUSTOMER ACCOUNT PROMPT')
	A																		
	A																		CA01(15 'END OF PROGRAM')
	A																		PUTRETAIN OVERLAY
	A																1	3	'ACCOUNT MASTER UPDATE'
	A																3	3	'FROM ACCOUNT NUMBER'
	A										ACCTFROM		5	Y	0	I	3	23	CHECK(ME)
	A		99																ERRMSG('INVALID FROM ACCOUNT +
	A																		NUMBER' 99)
	A		98																ERRMSG('INSUFFICIENT FUNDS IN FROM +
	A																		ACCOUNT' 98)
	A																4	3	'TO ACCOUNT NUMBER'
	A										ACCTTO		5	Y	0	I	4	23	CHECK(ME)
	A		97																ERRMSG('INVALID TO ACCOUNT +
	A																		NUMBER' 97)
	A																5	3	'AMOUNT TRANSFERRED'
	A										TRANSAMT		10	Y	02	I	5	23	
	A								R		ERRFMT								
	A		96														6	5	'INVALID FILE STATUS'
	A		96														7	5	'INVALID KEY IN REWRITE'
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		

Figure 63 (Part 1 of 5). Example of Use of Commitment Control

```
5763CB1                              COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.     ACCOUNT.
    3  000300   AUTHOR.       PROGRAMMER NAME.
    4  000400   INSTALLATION. TORONTO COBOL DEVELOPMENT CENTRE.
    5  000500   DATE-WRITTEN. 08/31/88.
    6  000600   DATE-COMPILED.  08/31/88 11:04:46   .
    7  000700 ENVIRONMENT DIVISION.
    8  000800 CONFIGURATION SECTION.
    9  000900 SOURCE-COMPUTER. IBM-S38.
   10  001000 OBJECT-COMPUTER. IBM-S38.
   11  001100 INPUT-OUTPUT SECTION.
   12  001200 FILE-CONTROL.
   13  001300     SELECT ACCOUNT-FILE  ASSIGN TO DATABASE-ACCTMST
   14  001400         ORGANIZATION IS INDEXED
   15  001500         ACCESS IS DYNAMIC
   16  001600         RECORD IS EXTERNALLY-DESCRIBED-KEY
   17  001700         FILE STATUS IS ACCOUNT-FILE-STATUS.
   18  001800     SELECT DISPLAY-FILE ASSIGN TO WORKSTATION-ACCTFMTS-SI 1
   19  001900         ORGANIZATION IS TRANSACTION.
       002000*****************************************************************
   20  002100 I-O-CONTROL.
   21  002200     COMMITMENT CONTROL FOR ACCOUNT-FILE. 2
       002300*****************************************************************
   22  002400 DATA DIVISION.
   23  002500 FILE SECTION.
   24  002600 FD  ACCOUNT-FILE
   25  002700     LABEL RECORDS ARE STANDARD.
   26  002800 01  ACCOUNT-RECORD.
   27  002900     COPY DDS-ALL-FORMATS OF ACCTMST.
   28 +000001       05  ACCTMST-RECORD PIC X(82).                          <-ALL-FMTS
      +000002*    I-O FORMAT:ACCNTREC   FROM FILE ACCTMST    OF LIBRARY COB38EX  <-ALL-FMTS
      +000003*                                                               <-ALL-FMTS
      +000004*THE KEY DEFINITIONS FOR RECORD FORMAT  ACCNTREC                 <-ALL-FMTS
      +000005*  NUMBER              NAME              RETRIEVAL     TYPE    ALTSEQ  <-ALL-FMTS
      +000006*   0001   ACCNTKEY                       ASCENDING   SIGNED     NO   <-ALL-FMTS
   29 +000007       05  ACCNTREC      REDEFINES ACCTMST-RECORD.              <-ALL-FMTS
   30 +000008           06 ACCNTKEY          PIC S9(5).                     <-ALL-FMTS
   31 +000009           06 NAME              PIC X(20).                     <-ALL-FMTS
   32 +000010           06 ADDR              PIC X(20).                     <-ALL-FMTS
   33 +000011           06 CITY              PIC X(20).                     <-ALL-FMTS
   34 +000012           06 STATE             PIC X(2).                      <-ALL-FMTS
   35 +000013           06 ZIP               PIC S9(5).                     <-ALL-FMTS
   36 +000014           06 BALANCE           PIC S9(8)V9(2).                <-ALL-FMTS
   37  003000
   38  003100 FD  DISPLAY-FILE
   39  003200     LABEL RECORDS ARE STANDARD.
   40  003300 01  DISPLAY-REC.
   41  003400     COPY DDS-ALL-FORMATS OF ACCTFMTS.
   42 +000001       05  ACCTFMTS-RECORD PIC X(20).                         <-ALL-FMTS
      +000002*  INPUT FORMAT:ACCTPMT    FROM FILE ACCTFMTS   OF LIBRARY COB38EX  <-ALL-FMTS
      +000003*                          CUSTOMER ACCOUNT PROMPT                <-ALL-FMTS
   43 +000004       05  ACCTPMT-I     REDEFINES ACCTFMTS-RECORD.             <-ALL-FMTS
   44 +000005           06 ACCTFROM          PIC S9(5).                     <-ALL-FMTS
```

Figure 63 (Part 2 of 5). Example of Use of Commitment Control

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   45 +000006           06 ACCTTO            PIC S9(5).                                   <-ALL-FMTS
   46 +000007           06 TRANSAMT          PIC S9(8)V9(2).                              <-ALL-FMTS
      +000008* OUTPUT FORMAT:ACCTPMT    FROM FILE ACCTFMTS   OF LIBRARY COB38EX           <-ALL-FMTS
      +000009*                          CUSTOMER ACCOUNT PROMPT                           <-ALL-FMTS
      +000010*      05  ACCTPMT-O     REDEFINES ACCTFMTS-RECORD.                        <-ALL-FMTS
      +000011*  INPUT FORMAT:ERRFMT     FROM FILE ACCTFMTS   OF LIBRARY COB38EX           <-ALL-FMTS
      +000012*                                                                            <-ALL-FMTS
      +000013*      05  ERRFMT-I      REDEFINES ACCTFMTS-RECORD.                        <-ALL-FMTS
      +000014* OUTPUT FORMAT:ERRFMT     FROM FILE ACCTFMTS   OF LIBRARY COB38EX           <-ALL-FMTS
      +000015*                                                                            <-ALL-FMTS
      +000016*      05  ERRFMT-O      REDEFINES ACCTFMTS-RECORD.                        <-ALL-FMTS
   47  003500 WORKING-STORAGE SECTION.
   48  003600 77  ACCOUNT-FILE-STATUS           PIC X(2).
   49  003700 77  IND-ON                       PIC 1    VALUE B"1".
   50  003800 77  IND-OFF                      PIC 1    VALUE B"0".
   51  003900 01  DISPFILE-INDICS.
   52  004000     COPY DDS-ALL-FORMATS-INDIC OF ACCTFMTS. 3
   53 +000001       05  ACCTFMTS-RECORD.                                                 <-ALL-FMTS
      +000002*  INPUT FORMAT:ACCTPMT    FROM FILE ACCTFMTS   OF LIBRARY COB38EX           <-ALL-FMTS
      +000003*                          CUSTOMER ACCOUNT PROMPT                           <-ALL-FMTS
   54 +000004           06 ACCTPMT-I-INDIC.                                               <-ALL-FMTS
   55 +000005                07 IN15         PIC 1  INDIC 15.                           <-ALL-FMTS
      +000006*                          END OF PROGRAM                                    <-ALL-FMTS
   56 +000007                07 IN97         PIC 1  INDIC 97.                           <-ALL-FMTS
      +000008*                          INVALID TO ACCOUNT NUMBER                         <-ALL-FMTS
   57 +000009                07 IN98         PIC 1  INDIC 98.                           <-ALL-FMTS
      +000010*                          INSUFFICIENT FUNDS IN FROMACCOUNT                 <-ALL-FMTS
   58 +000011                07 IN99         PIC 1  INDIC 99.                           <-ALL-FMTS
      +000012*                          INVALID FROM ACCOUNT NUMBER                       <-ALL-FMTS
      +000013* OUTPUT FORMAT:ACCTPMT    FROM FILE ACCTFMTS   OF LIBRARY COB38EX           <-ALL-FMTS
      +000014*                          CUSTOMER ACCOUNT PROMPT                           <-ALL-FMTS
   59 +000015           06 ACCTPMT-O-INDIC.                                               <-ALL-FMTS
   60 +000016                07 IN97         PIC 1  INDIC 97.                           <-ALL-FMTS
      +000017*                          INVALID TO ACCOUNT NUMBER                         <-ALL-FMTS
   61 +000018                07 IN98         PIC 1  INDIC 98.                           <-ALL-FMTS
      +000019*                          INSUFFICIENT FUNDS IN FROMACCOUNT                 <-ALL-FMTS
   62 +000020                07 IN99         PIC 1  INDIC 99.                           <-ALL-FMTS
      +000021*                          INVALID FROM ACCOUNT NUMBER                       <-ALL-FMTS
      +000022*  INPUT FORMAT:ERRFMT     FROM FILE ACCTFMTS   OF LIBRARY COB38EX           <-ALL-FMTS
      +000023*                                                                            <-ALL-FMTS
      +000024*          06 ERRFMT-I-INDIC.                                                <-ALL-FMTS
      +000025* OUTPUT FORMAT:ERRFMT     FROM FILE ACCTFMTS   OF LIBRARY COB38EX           <-ALL-FMTS
      +000026*                                                                            <-ALL-FMTS
   63 +000027           06 ERRFMT-O-INDIC.                                                <-ALL-FMTS
   64 +000028                07 IN95         PIC 1  INDIC 95.                           <-ALL-FMTS
   65 +000029                07 IN96         PIC 1  INDIC 96.                           <-ALL-FMTS
   66  004100
   67  004200 PROCEDURE DIVISION.
       004300 DECLARATIVES.
       004400 ERROR-SECTION SECTION.
       004500     USE AFTER STANDARD EXCEPTION PROCEDURE ON ACCOUNT-FILE.
       004600 ERROR-PARAGRAPH.
   68  004700     IF ACCOUNT-FILE-STATUS IS NOT EQUAL "23" THEN
   69  004800         MOVE IND-ON TO IN96 OF ERRFMT-O-INDIC       4
       004900     ELSE
   70  005000         MOVE IND-ON TO IN95 OF ERRFMT-O-INDIC. _____
   71  005100     WRITE DISPLAY-REC FORMAT IS "ERRFMT"
       005200           INDICATORS ARE ERRFMT-O-INDIC.
```

Figure 63 (Part 3 of 5). Example of Use of Commitment Control

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   72  005300     READ DISPLAY-FILE.
   73  005400     CLOSE DISPLAY-FILE
       005500           ACCOUNT-FILE.
   74  005600     STOP RUN.
       005700 END DECLARATIVES.
       005800 MAIN-PROGRAM SECTION.
       005900 MAINLINE.
   75  006000     OPEN I-O DISPLAY-FILE
       006100          I-O ACCOUNT-FILE.
   76  006200     MOVE ZEROS TO ACCTPMT-I-INDIC
       006300                   ACCTPMT-O-INDIC.
   77  006400     PERFORM WRITE-READ-DISPLAY.
   78  006500     PERFORM VERIFY-ACCOUNT-NO UNTIL IN15 EQUAL IND-ON.
   79  006600     CLOSE DISPLAY-FILE
       006700           ACCOUNT-FILE.
   80  006800     STOP RUN.
       006900 VERIFY-ACCOUNT-NO.
   81  007000     PERFORM VERIFY-TO-ACCOUNT.
   82  007100     IF IN97 OF ACCTPMT-O-INDIC EQUAL IND-OFF THEN
   83  007200         PERFORM VERIFY-FROM-ACCOUNT.
   84  007300     PERFORM WRITE-READ-DISPLAY.
       007400 VERIFY-FROM-ACCOUNT.
   85  007500     MOVE ACCTFROM TO ACCNTKEY.
   86  007600     READ ACCOUNT-FILE
   87  007700         INVALID KEY MOVE IND-ON TO IN99 OF ACCTPMT-O-INDIC.
   88  007800     IF IN99 OF ACCTPMT-O-INDIC EQUAL IND-ON THEN 6
       007900*                                              *
       008000         ROLLBACK
   89  008100*                                              *
       008200     ELSE
   90  008300         PERFORM UPDATE-FROM-ACCOUNT.
       008400 VERIFY-TO-ACCOUNT.
   91  008500     MOVE ACCTTO TO ACCNTKEY.
   92  008600     READ ACCOUNT-FILE
   93  008700         INVALID KEY MOVE IND-ON TO IN97 OF ACCTPMT-O-INDIC. 7
   94  008800     IF IN97 OF ACCTPMT-O-INDIC EQUAL IND-ON THEN
       008900*                                              *
       009000         ROLLBACK 8
   95  009100*                                              *
       009200     ELSE
   96  009300         PERFORM UPDATE-TO-ACCOUNT.
       009400 UPDATE-TO-ACCOUNT.
   97  009500     ADD TRANSAMT TO BALANCE.
   98  009600     REWRITE ACCOUNT-RECORD.
       009700 UPDATE-FROM-ACCOUNT.
   99  009800     SUBTRACT TRANSAMT FROM BALANCE.
  100  009900     REWRITE ACCOUNT-RECORD.
  101  010000     IF BALANCE IS LESS THAN 0 THEN
  102  010100         MOVE IND-ON TO IN98 OF ACCTPMT-O-INDIC
       010200*                                              *
       010300         ROLLBACK _________
  103  010400*                                              *
```

Figure 63 (Part 4 of 5). Example of Use of Commitment Control

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
      010500     ELSE
      010600*                                              *
      010700         COMMIT.  10
  104 010800*                                              *
      010900 WRITE-READ-DISPLAY.
  105 011000     WRITE DISPLAY-REC FORMAT IS "ACCTPMT"
      011100         INDICATORS ARE ACCTPMT-O-INDIC. C/xx)
  106 011200     MOVE ZEROS TO ACCTPMT-I-INDIC
      011300                   ACCTPMT-O-INDIC.
  107 011400     READ DISPLAY-FILE RECORD
      011500         INDICATORS ARE ACCTPMT-I-INDIC.
      011600
                        * * * * *  E N D  O F  S O U R C E  * * * * *
```

1 A separate indicator area is provided for the program.

2 The COMMITMENT CONTROL clause specifies files to be placed under commitment control. Any files named in this clause are affected by the COMMIT and ROLLBACK verbs.

3 The Format 2 COPY statement with the indicator attribute INDIC, defines data description entries in WORKING-STORAGE for the indicators to be used in the program.

4 IN96 is set if there is an invalid file status.

5 IN95 is set if there is an INVALID KEY condition on the REWRITE operation.

6 IN99 is set if the entered account number is invalid for the account to which money is being transferred.

7 IN97 is set if the entered account number is invalid for the account to which money is being transferred.

8 If an INVALID KEY condition occurs on the READ, a ROLLBACK is used and the record lock placed on the record after the first READ is released.

9 If the transfer of funds is invalid (an indicator has been set), the ROLLBACK statement is processed. All changes made to data base files under commitment control are cancelled.

10 If the transfer of funds was valid (no indicators have been set), the COMMIT statement is processed, and all changes made to data base files under commitment control then become permanent.

11 The INDICATORS phrase is required for options on the work station screen that are controlled by indicators.

Figure 63 (Part 5 of 5). Example of Use of Commitment Control

Performance Considerations

Segmentation

Use of segmentation increases the compile and run times of the COBOL program. The segmentation feature is provided only for compatibility with other systems. It is not necessary to be concerned with storage management when using AS/400 COBOL programs.

Debugging

COBOL source language debugging is provided to help the COBOL programmer debug a program that is not functioning as expected. Use of this facility increases the compile and run times of a COBOL program.

*NORANGE Option

This option of the GENOPT parameter of the CRTCBLPGM command removes the run-time checks for subscript ranges. If frequent references to tables are made and the subscripts always reference elements within the table, use of this option can improve performance.

Indicators

If you use indicators in a separate indicator area (INDARA keyword specified in DDS) instead of in the record area, the use of the OCCURS clause to specify a table with up to 99 indicators can improve performance.

Commitment Control

Generally, the use of commitment control increases the run time of a COBOL program. In addition, the record locking which results from the use of commitment control by a job may cause delays for other users attempting to access the same file.

Program Loops

When a program repeatedly processes the same series of instructions, and it is apparent that this will continue indefinitely, the program is in a loop. To identify loops, you can use information known about the program itself, as follows:

- Time: If the actual run time is substantially exceeding the expected run time, the program could be in a loop.
- I-O operations: If no input/output operations are taking place and I-O is expected to be occurring repeatedly, the program is probably in a loop.

Tracing a Loop in a Program

Frequently, a loop encompasses many instructions in a program. In this case, you can use the COBOL debugging features as described in Chapter 11, “Using the Additional COBOL Functions,” or the AS/400 debugging capabilities of Trace as described in “Debugging Your Program”

Errors That Can Cause a Loop

A PERFORM statement with an UNTIL clause can cause a loop when the condition specified in the UNTIL clause cannot be met. For example:

```
PERFORM ... UNTIL COUNTR LESS THAN ZERO
```

where COUNTR is an unsigned numeric item.

A GO TO statement that refers to a previous procedure-name can cause a loop when no conditional statement exists to prevent the GO TO statement from being processed again. For example:

```
PARA-1.
  MOVE ...
  MOVE ...
  MOVE ...
PARA-2.
  MOVE ...
  GO TO PARA-1.
```

A possible variation of this case is when a conditional statement exists, but the condition cannot be met or the statement does not branch (through a GO TO statement) to a paragraph outside the range of the loop.

Exceptions and Some of Their Causes

MCH1202 data exception:

- A numeric elementary item has been used as a source when no valid data has been previously stored in it. The item should have a VALUE clause, or a MOVE statement should be used to initialize its value.
- An attempt has been made to place nonnumeric data in a numeric item.
- Bad data was written to a subfile earlier in the program. The subfile data is not validated until it is written to the screen, so the 1202 error can occur on the WRITE of a subfile control record, but the bad data was actually put to the subfile earlier.

MCH3601 invalid pointer:

- A reference is made to a record or a field within a record and the associated file has been closed or has never been opened.

  For example, the OPEN for the file was unsuccessful and the processing of any other I-O statement for that file is attempted. The file status should be checked before any other I-O is attempted.

CPF2415 end of request:

- An attempt has been made to accept input from the job input stream while the system is running in batch mode and no input is available.

Recovery after a Failure

Recovery with Commitment Control

When the system is restarted after a failure, files under commitment control are automatically restored to their status at the last commitment boundary. For additional information about commitment control see “Commitment Control Considerations” on page 247.

For a job failure (either because of user or system error), files under commitment control are restored as part of job termination to the files’ status at the previous commitment boundary.

Because files under commitment control are rolled back after system or process failure, this feature may be used to aid in restarting. You can create a separate record to store data which may be useful should it become necessary to restart a job. This restart data can include items such as totals, counters, record key values, relative key values, and other relevant processing information from an application.

By having the above restart data in a file under commitment control, that data too will be permanently stored in the data base when a `COMMIT` statement is issued. When a `ROLLBACK` occurs after job or process failure, you may retrieve a record of the extent of processing successfully processed before failure. Note that the above method is only a suggested programming technique and will not always be suitable, depending on the application.

Transaction File Recovery

In some cases, you can recover from I-O errors on `TRANSACTION` files without intervention by the operator, or the varying off/varying on of work stations or communications devices.

For potentially recoverable I-O errors on `TRANSACTION` files, the system initiates action in addition to the steps that must be taken in the application program to attempt error recovery.

By examining the file status after an I-O operation, the application program can determine whether a recovery from an I-O error on the `TRANSACTION` file may be possible. If the File Status Key has a value of `9N`, the application program may be able to recover from the I-O error. A recovery procedure must be coded as part of the application program and varies depending on whether a single device was acquired by the `TRANSACTION` file or whether there were multiple devices attached.

For a file with one acquired device:

1. Close the `TRANSACTION` file with the I-O error.
2. Reopen the file.
3. Process the steps necessary to retry the failing I-O operation. This may involve a number of steps, depending on the type of program device used. (For example, if the last I-O operation was a `READ`, you may have to repeat one or more `WRITE` statements, which were processed prior to the `READ` statement.) For more information on recovery procedures, see the *Communications Management Guide* and the *System/38 Data Communications Programmer's Guide*.

For a file with multiple devices acquired:

1. DROP the program device that caused the I-O error on the TRANSACTION file.
2. ACQUIRE the same program device.
3. See Step 3 above.

Application program recovery attempts should typically be tried only once.

If the recovery attempt fails:

- If the file has only one program device attached, terminate the program through processing of the STOP RUN or EXIT PROGRAM statement, and attempt to locate the source of the error.
- If the file has multiple acquired program devices, you may wish to do one of the following:
  - Continue processing without the program device that caused the I-O error on the TRANSACTION file, and reacquire the device later.
  - Terminate the program.

For a description of major/minor return codes that may aid in diagnosing I-O errors on the TRANSACTION file, see the *Data Management Guide*.

Figure 64 gives an example of an error recovery procedure.

IBM International Business Machines — DATA DESCRIPTION SPECIFICATIONS — GX21-7784-1 UM/060* Printed in U.S.A. *Number of sheets per pad may vary slightly.

File		Keying Instruction	Graphic							Description	Page of
Programmer	Date		Key								

A

Sequence Number	Form Type	And/Or/Comment (A/O/*)	Conditioning: Not (N)	Condition Name: Indicator	Not (N)	Indicator	Not (N)	Indicator	Name Type (R/P/K/S/C/E/O)	Reserved	Name	Reference (R)	Length	Data Type (G/A/P/B/S/B A/B/X/Y/N/I/W)	Decimal Positions	Usage (B/D/I/P/B/H/M)	Location: Line	Pos	Functions
	A	*		DISPLAY					FILE		FOR ERROR RECOVERY								EXAMPLE
	A	*																	
	A																		INDARA
	A								R		FORMAT1								CF01(01 'END OF PROGRAM')
	A	*																	
	A																12	28	'ENTER INPUT '
	A										INPUTFLD		5			I	12	42	
	A																28	26	'F1 = TERMINATE'
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		
	A																		

Figure 64 (Part 1 of 4). Example of Error Recovery Procedure

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
    1  000100 IDENTIFICATION DIVISION.
    2  000200 PROGRAM-ID.  RECOVERY.
    3  000300 ENVIRONMENT DIVISION.
    4  000400 CONFIGURATION SECTION.
    5  000500 SOURCE-COMPUTER. IBM-S38.
    6  000600 OBJECT-COMPUTER. IBM-S38.
    7  000700 INPUT-OUTPUT SECTION.
    8  000800 FILE-CONTROL.
    9  000900     SELECT RECOVFILE
   10  001000         ASSIGN TO WORKSTATION-RECOVFILE-SI
   11  001100         ORGANIZATION IS TRANSACTION
   12  001200         ACCESS MODE IS SEQUENTIAL
   13  001300         FILE STATUS IS STATUS-FLD, STATUS-FLD-2
   14  001400         CONTROL-AREA IS CONTROL-FLD.
   15  001500     SELECT PRINTER-FILE
   16  001600         ASSIGN TO PRINTER-QPRINT.
   17  001700
   18  001800 DATA DIVISION.
   19  001900 FILE SECTION.
   20  002000 FD  RECOVFILE
   21  002100     LABEL RECORDS ARE OMITTED
   22  002200     DATA RECORD IS RECOV-REC.
   23  002300 01  RECOV-REC.
   24  002400     COPY DDS-ALL-FORMATS OF RECOVFILE.
   25 +000001       05  RECOVFILE-RECORD PIC X(5).                                    <-ALL-FMTS
      +000002*  INPUT FORMAT:FORMAT1    FROM FILE RECOVFILE  OF LIBRARY EXMPLIB      <-ALL-FMTS
      +000003*                                                                       <-ALL-FMTS
   26 +000004       05  FORMAT1-I     REDEFINES RECOVFILE-RECORD.                    <-ALL-FMTS
   27 +000005           06 INPUTFLD          PIC X(5).                              <-ALL-FMTS
      +000006* OUTPUT FORMAT:FORMAT1    FROM FILE RECOVFILE  OF LIBRARY EXMPLIB      <-ALL-FMTS
      +000007*                                                                       <-ALL-FMTS
      +000008*      05  FORMAT1-O     REDEFINES RECOVFILE-RECORD.                    <-ALL-FMTS
   28  002500
   29  002600 FD  PRINTER-FILE.
   30  002700 01  PRINTER-REC.
   31  002800    05  PRINTER-RECORD            PIC X(132).
   32  002900
   33  003000 WORKING-STORAGE SECTION.
   34  003100
   35  003200 01  I-O-VERB                     PIC X(10).
   36  003300 01  STATUS-FLD                   PIC X(2).
   37  003400    88  NO-ERROR                    VALUE "00".
   38  003500    88  ACQUIRE-FAILED              VALUE "9H".
   39  003600    88  TEMPORARY-ERROR             VALUE "9N".
   40  003700 01  STATUS-FLD-2                 PIC X(4).
   41  003800 01  CONTROL-FLD.
   42  003900    05  FUNCTION-KEY              PIC X(2).
   43  004000    05  PGM-DEVICE-NAME           PIC X(10).
   44  004100    05  RECORD-FORMAT             PIC X(10).
   45  004200 01  END-INDICATOR                PIC 1   INDICATOR 1
   46  004300                                    VALUE B"0".
   47  004400    88  END-NOT-REQUESTED           VALUE B"0".
   48  004500    88  END-REQUESTED               VALUE B"1".
   49  004600 01  USE-PROC-FLAG                PIC 1
   50  004700                                    VALUE B"0".
   51  004800    88  USE-PROC-NOT-EXECUTED       VALUE B"0".
   52  004900    88  USE-PROC-EXECUTED           VALUE B"1".
   53  005000 01  RECOVERY-FLAG                PIC 1
   54  005100                                    VALUE B"0".
   55  005200    88  NO-RECOVERY-DONE            VALUE B"0".
   56  005300    88  RECOVERY-DONE               VALUE B"1".
```

Figure 64 (Part 2 of 4). Example of Error Recovery Procedure

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
   57  005400 01  HEADER-LINE.
   58  005500    05  FILLER                    PIC X(60)
   59  005600                                    VALUE SPACES.
   60  005700    05  FILLER                    PIC X(72)
   61  005800                                    VALUE "ERROR REPORT".
   62  005900 01  DETAIL-LINE.
   63  006000    05  FILLER                    PIC X(15)
   64  006100                                    VALUE SPACES.
   65  006200    05  DESCRIPTION               PIC X(25)
   66  006300                                    VALUE SPACES.
   67  006400    05  DETAIL-VALUE              PIC X(92)
   68  006500                                    VALUE SPACES.
   69  006600 01  MESSAGE-LINE.
   70  006700    05  FILLER                    PIC X(15)
   71  006800                                    VALUE SPACES.
   72  006900    05  DESCRIPTION               PIC X(117)
   73  007000                                    VALUE SPACES.
   74  007100 PROCEDURE DIVISION.
       007200 DECLARATIVES.
       007300 HANDLE-ERRORS SECTION.
       007400     USE AFTER STANDARD ERROR PROCEDURE ON RECOVFILE.  1
       007500 DISPLAY-ERROR.
   75  007600     SET USE-PROC-EXECUTED TO TRUE.
   76  007700     WRITE PRINTER-REC FROM HEADER-LINE AFTER ADVANCING PAGE.
   77  007800     MOVE "ERROR OCCURED IN" TO DESCRIPTION OF DETAIL-LINE.
   78  007900     MOVE I-O-VERB TO DETAIL-VALUE OF DETAIL-LINE.
   79  008000     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 5 LINES.
   80  008100     MOVE "FILE STATUS =" TO DESCRIPTION OF DETAIL-LINE.
   81  008200  2    MOVE STATUS-FLD TO DETAIL-VALUE OF DETAIL-LINE.
   82  008300     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
   83  008400     MOVE "EXTENDED FILE STATUS =" TO DESCRIPTION OF DETAIL-LINE.
   84  008500     MOVE STATUS-FLD-2 TO DETAIL-VALUE OF DETAIL-LINE.
   85  008600     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
   86  008700     MOVE "CONTROL-AREA =" TO DESCRIPTION OF DETAIL-LINE.
   87  008800     MOVE CONTROL-FLD TO DETAIL-VALUE OF DETAIL-LINE.
   88  008900     WRITE PRINTER-REC FROM DETAIL-LINE AFTER ADVANCING 2 LINES.
       009000 CHECK-ERROR.
   89  009100     IF TEMPORARY-ERROR AND NO-RECOVERY-DONE THEN
   90  009200         MOVE "***ERROR RECOVERY BEING ATTEMPTED***"  3
       009300             TO  DESCRIPTION OF MESSAGE-LINE
   91  009400         WRITE PRINTER-REC FROM MESSAGE-LINE
       009500             AFTER ADVANCING 3 LINES
   92  009600         PERFORM ERROR-RECOVERY
       009700     ELSE
   93  009800  4        IF RECOVERY-DONE THEN
   94  009900             MOVE "***ERROR AROSE FROM RETRY AFTER RECOVERY***"
       010000                 TO  DESCRIPTION OF MESSAGE-LINE
   95  010100             WRITE PRINTER-REC FROM MESSAGE-LINE
       010200                 AFTER ADVANCING 3 LINES
   96  010300             MOVE "***PROGRAM TERMINATED***"
       010400                 TO  DESCRIPTION OF MESSAGE-LINE
   97  010500             WRITE PRINTER-REC FROM MESSAGE-LINE
       010600                 AFTER ADVANCING 2 LINES
   98  010700             GO TO ERROR-EXIT
       010800         ELSE
   99  010900             SET NO-RECOVERY-DONE TO TRUE.
  100  011000     MOVE "***EXECUTION CONTINUES***"
       011100         TO DESCRIPTION OF MESSAGE-LINE.
  101  011200     WRITE PRINTER-REC FROM MESSAGE-LINE
       011300         AFTER ADVANCING 2 LINES.
```

Figure 64 (Part 3 of 4). Example of Error Recovery Procedure

```
5763CB1                            COBOL SOURCE LISTING
 STMT SEQNBR -A 1 B..+....2....+....3....+....4....+....5....+....6....+....7..IDENTFCN S  COPYNAME   CHG/DATE
  102  011400     GO TO END-OF-DECLARATIVES.
       011500 ERROR-RECOVERY.
  103  011600     SET RECOVERY-DONE TO TRUE.
  104  011700     DROP PGM-DEVICE-NAME FROM RECOVFILE.
  105  011800     ACQUIRE PGM-DEVICE-NAME FOR RECOVFILE. ____
       011900 ERROR-EXIT.
  106  012000     CLOSE RECOVFILE
       012100           PRINTER-FILE.
       012200 END-OF-DECLARATIVES.
       012300 END DECLARATIVES.
       012400
       012500 MAIN-PROGRAM SECTION.
       012600 MAINLINE.
  107  012700     MOVE "OPEN" TO I-O-VERB.
  108  012800     OPEN I-O     RECOVFILE
       012900          OUTPUT  PRINTER-FILE.
  109  013000     PERFORM I-O-PARAGRAPH UNTIL END-REQUESTED.  6
  110  013100     CLOSE RECOVFILE
       013200           PRINTER-FILE.
  111  013300     STOP RUN.
       013400 I-O-PARAGRAPH.
  112  013500     MOVE "WRITE" TO I-O-VERB.
  113  013600     SET USE-PROC-NOT-EXECUTED TO TRUE.
  114  013700     WRITE RECOV-REC FORMAT IS "FORMAT1"
       013800           INDICATOR IS END-INDICATOR.
  115  013900     IF USE-PROC-EXECUTED AND RECOVERY-DONE THEN  7
  116  014000         GO TO I-O-PARAGRAPH.
  117  014100     MOVE "READ" TO I-O-VERB.
  118  014200     SET USE-PROC-NOT-EXECUTED TO TRUE.
  119  014300     SET NO-RECOVERY-DONE TO TRUE.
  120  014400     READ RECOVFILE FORMAT IS "FORMAT1"
       014500         INDICATOR IS END-INDICATOR.  8
  121  014600     IF NO-ERROR THEN
  122  014700         PERFORM SOME-PROCESSING.
       014800 SOME-PROCESSING.
  123  014900*    (INSERT SOME DATABASE PROCESSING, FOR EXAMPLE)
                          * * * * *  E N D  O F  S O U R C E  * * * * *
```

1 This defines processing that takes place when an I-O error occurs on `RECOVFILE`.

2 This prints out information to help in diagnosing the problem.

3 If the file-status equals `9N` (temporary error), and no previous error recovery has been attempted for this I-O operation, error recovery is now attempted.

4 If recovery was previously attempted, it is not attempted now. This is done to avoid program looping.

5 Recovery consists of dropping, then reacquiring, the program device on which the I-O error occurred.

6 The mainline of the program consists of writing to and reading from a device until the user signals an end to the program by pressing `F1`.

7 If the `WRITE` operation failed but recovery was done, the `WRITE` is attempted again.

8 If the `READ` operation failed, processing will continue by writing to the device again, and then attempting the `READ` again.

Figure 64 (Part 4 of 4). Example of Error Recovery Procedure

Inter-Program Communication Considerations

The AS/400 system allows inter-program communication between COBOL and COBOL or non-COBOL programs. For the following discussion a main program is defined as the COBOL program that is highest in the program stack. The main program is the first program in the run unit. A run unit is defined as a set of one or more programs that functions as a unit at run time to provide a problem solution. A run unit starts with the first COBOL program in the program stack and includes all programs (of any type) that are below it in the program stack. A subprogram is a program in the run unit below the main program in the program stack.

Return of Control From a Called Program

In COBOL, you can issue either a `STOP RUN` statement or an `EXIT PROGRAM` statement to return control from a called program. The action of these statements depends upon the run time environment, as explained below.

- **Exit Program**
  - When issued from a main program, control passes to the next statement (no operation is processed).
  - When issued from a subprogram, control returns to the calling program.
- **Stop Run**
  - Whether issued from a main program or a subprogram, the run unit is terminated. Control returns to the (non-COBOL) program that called the main program.

Initialization of Storage

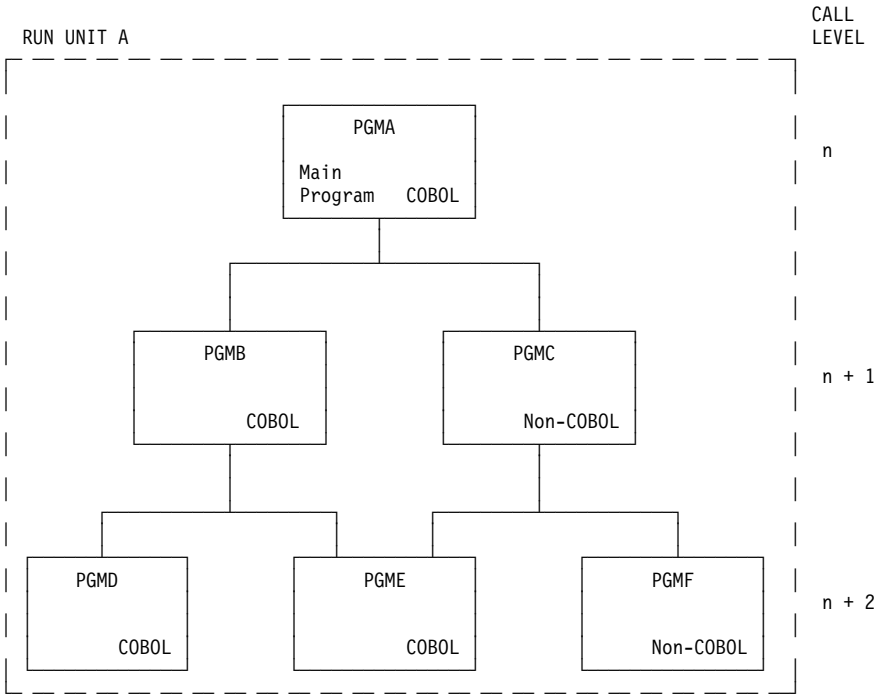
The first time a COBOL program in a run unit is called, its storage is initialized. Storage is initialized again under the following conditions:

- The run unit is terminated, then reinitiated.
- The program is cancelled (COBOL `CANCEL` statement, RPG/400 `FREE` operation, `CL` command `RCLRSC`), then called again.

If a non-COBOL program is named in a `CANCEL` statement, its name must conform to the rules for formation of a COBOL program name.

The following examples illustrate the use of the `EXIT PROGRAM` and `STOP RUN` statements in different parts of a run unit.

- The example in Figure 65 on page 263 shows a single run unit.
- The example in Figure 66 on page 264 shows multiple run units.
- The example in Figure 67 on page 265 shows a run unit with a shared program that is both a subprogram and a main program.



PROGRAM PROCESSING STATEMENT

STATEMENT	PGMA	PGMB	PGMD	PGME
EXIT PROGRAM	1	2	2	2
STOP RUN	3	3	3	3

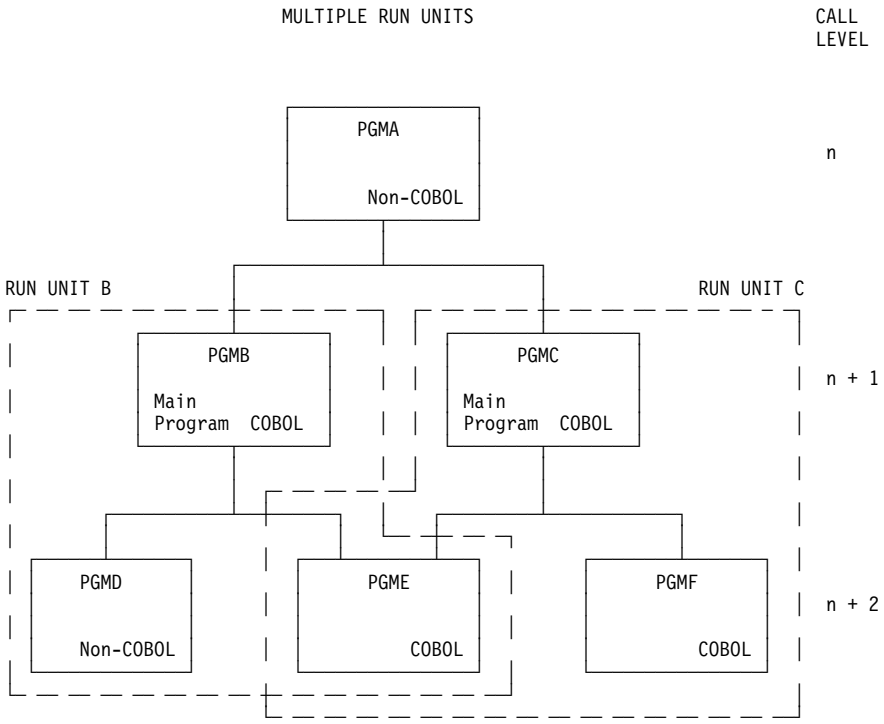
Figure 65. Example of a Single Run Unit

1 No operation is processed because the statement is processed in a main program. Processing continues with the next statement in `PGMA`.

2 Control returns to the caller of the program that processes the `EXIT PROGRAM` statement.

3 Run unit A terminates. For all programs in the run unit, open files are closed. Storage is freed for all programs in the run unit. Control returns to the program that is at call level n-1. If n=1, the following considerations apply:

- Run unit A operates as a routing step. See the *CL Reference* for more information.
- For batch jobs, the `STOP RUN` causes the job to end. For interactive jobs, control returns to the system and the system terminates the routing step.



	PROGRAM PROCESSING STATEMENT				
STATEMENT	PGMB	PGMC	PGME (RUN UNIT B)	PGME (RUN UNIT C)	PGMF
EXIT PROGRAM	1	1	2	2	2
STOP RUN	3	4	3	4	4

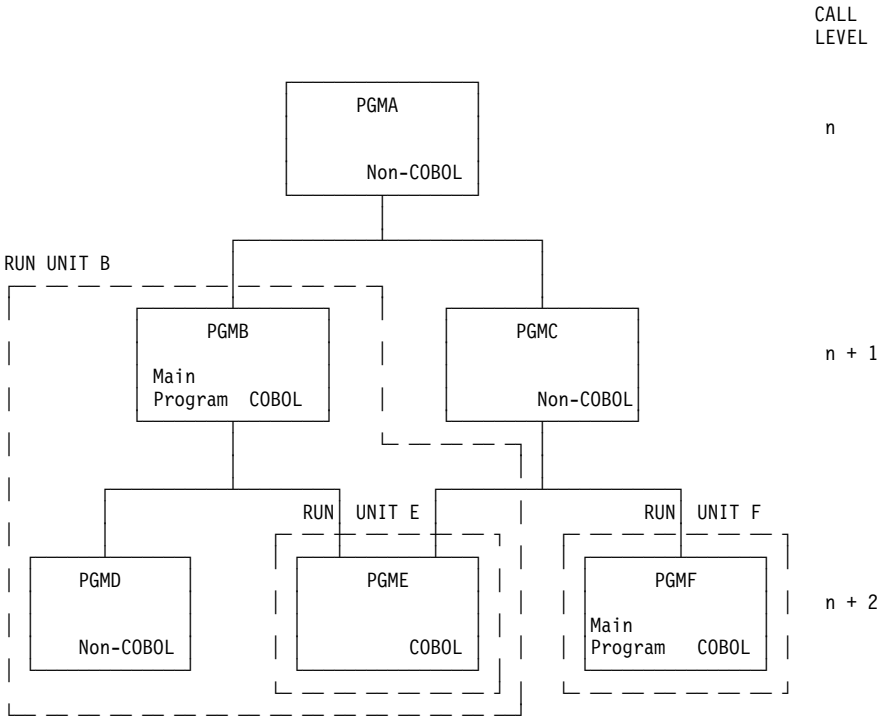
Figure 66. Example of Multiple Run Units

1 No operation is processed because the statement is processed in a main program. Processing continues with the next statement in the main program.

2 Control returns to the caller of the program that processes the `EXIT PROGRAM` statement.

3 Run unit B terminates. All open files in run unit B are closed. Storage is freed for all programs in run unit B. Control returns to the caller of the main program for the run unit (`PGMA`).

4 Run unit C terminates. All open files in run unit C are closed. Storage is freed for all programs in run unit C. Control returns to the caller of the main program for the run unit (`PGMA`).



	PROGRAM PROCESSING STATEMENT			
STATEMENT	PGMB	PGME (RUN UNIT B)	PGME (RUN UNIT E)	PGMF
EXIT PROGRAM	1	2	1	1
STOP RUN	3	3	4	5

Figure 67. Example of a Run Unit with a Shared Program

1 No operation is processed because the statement is processed in a main program. Processing continues with the next statement in the main program.

2 Control returns to the caller of the program that processes the `EXIT PROGRAM` statement.

3 Run unit B terminates. All open files in run unit B are closed. Storage is freed for all programs in run unit B. Control returns to the caller of the main program for the run unit (`PGMA`).

4 Run unit E terminates. All open files in run unit E are closed. Storage is freed for `PGME`. Control returns to the caller of the main program for the run unit (`PGMC`).

5 Run unit F terminates. All open files in run unit F are closed. Storage is freed for `PGMF`. Control returns to the caller of the main program for the run unit (`PGMC`).

Local Data Area

The system automatically creates a local data area for each job. The local data area is defined outside the COBOL program as an area of 1024 bytes of character data.

The local data area can be used to pass any desired information between programs in a job. This information may be free-form data, such as informal messages, or may consist of a fully structured or formatted set of fields.

When a job is submitted, the submitting job's local data area is copied into the submitted job's local data area. If there is no submitting job, the local data area is initialized to blanks.

A COBOL program can access the local data area for its job with the `ACCEPT` and `DISPLAY` statements, using a mnemonic name associated with the function-name `LOCAL-DATA`.

There is only one local data area associated with each job. If several work stations are acquired by a single job, still only one local data area exists for that job. There is *not* a local data area for each individual work station.

File Considerations

A file cannot be received as a parameter in a COBOL program. If a file is defined in both a calling program and a called program, it is treated as two separate files. The contents of the record area and the current record pointer in each program are independent, unless shared files are specified in CL commands.

The following statements affect file status differently:

- An `EXIT PROGRAM` statement does not change the status of any of the files in a run unit.
- A `STOP RUN` statement closes all the files in a run unit.
- A `CANCEL` statement does not change the status of any of the files in the program that is canceled. It does free the storage that contains information about the file. If the program has files that are open when the `CANCEL` statement is processed, those files remain open until the run unit is terminated. The program can no longer use the file. If the cancelled program is called again, the program considers the file closed. If the program opens the file, a new linkage to the file is established. This can cause additional system storage to be used.

Chapter 8. Identification and Environment Divisions

Note: Although card devices are not supported by System/38-Compatible COBOL, there are references to card devices in this chapter, which may help you when you are creating programs for System/38.

IDENTIFICATION DIVISION

The Identification Division must be the first division in every COBOL source program. This division names the source program and the object program. (A source program is the user-written COBOL program. An object program is the output from a compilation.)

The user may also include the date the program was written, the date of compilation, and other such documentary information about the program in the Identification Division.

Format

```
  IDENTIFICATION DIVISION.
  PROGRAM-ID. program-name.
[ AUTHOR. [ comment-entry ] . . . ]
[ INSTALLATION. [ comment-entry ] . . . ]
[ DATE-WRITTEN. [ comment-entry ] . . . ]
[ DATE-COMPILED. [ comment-entry ] . . . ]
[ SECURITY. [ comment-entry ] . . . ]
```

The Identification Division must begin with the words `IDENTIFICATION DIVISION` followed by a period and a space.

Coding Example

```
SEQUENCE        CONT  A    B                                          COBOL STATEMENT
PAGE  SERIAL
1   3  4   6    7     8    12   16   20   24   28   32   36   40   44
0 0 1  0 1 0          IDENTIFICATION  DIVISION.
       0 2 0          PROGRAM-ID.  SAMPLE.
       0 3 0          AUTHOR.  A  PROGRAMMER.
       0 4 0          INSTALLATION.  ROCHESTER  LAB.
       0 5 0          DATE  WRITTEN.  08/11/88.
       0 6 0          DATE-COMPILED.  08/12/88.
       0 7 0          SECURITY  NON-CONFIDENTIAL.
       0 8
```

PROGRAM-ID Paragraph

The first paragraph of the Identification Division must be the PROGRAM-ID paragraph. The PROGRAM-ID paragraph specifies the name by which the program is known to the system.

The name by which the program is known to the system can be overridden by the PGM parameter of the CRTCBLPGM command. See “Create COBOL Program Command” on page 37 for more information on the PGM parameter.

Program-name is a user-defined word that identifies the object program to the system. A program-name must include at least one alphabetic character. The system uses the first ten characters of program-name as the identifying name of the program; these first ten characters, therefore, should be a unique program-name.

The system expects the first character of program-name to be alphabetic; if it is numeric, it is converted as follows:

- 0 is converted to J
- 1 through 9 is converted to A through I.

The system does not include the hyphen as an allowable character; therefore, if any of the second through tenth characters are hyphens, they are converted to zeros.

To avoid such conversions, the user should not specify program-names with leading numerics or embedded hyphens.

Other Optional Paragraphs

The other paragraphs are optional; however, if they are written, they must appear in the order shown in the format.

The comment-entries serve only as documentation and do not affect the syntax of the program. The comment-entries in the optional paragraphs may be any combi-

nation of characters from the EBCDIC set and may be written in Area B on one or more lines. A hyphen is not permitted in the continuation area of Identification Division statements.

The DATE-COMPILED paragraph provides the compilation date of the source listing. When the comment-entry is specified, the entire entry is replaced with the current date. When the comment-entry is omitted, the compiler adds the current date to the DATE-COMPILED paragraph.

ENVIRONMENT DIVISION

The Environment Division, the second division of all COBOL source programs, identifies the following:

- The computer on which the source program is to be compiled
- The computer on which the object program is to be run
- The specific main storage size required to run the object program
- The linkage between the logical concept of the files and their records, and the physical aspects of the devices on which data is stored.

The Environment Division has two sections: the Configuration Section and the Input-Output Section.

The following shows the general format of the sections and paragraphs in the Environment Division, and defines the order of presentation in the source program.

Format

```
ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. source-computer-entry

OBJECT-COMPUTER. object-computer-entry

[ SPECIAL-NAMES. special-names-entry ]

[ INPUT-OUTPUT SECTION.

  FILE-CONTROL. { file-control-entry } . . .

[ I-O-CONTROL. input-output-control-entry ] ]
```

The Environment Division must begin with the words ENVIRONMENT DIVISION followed by a period and a space.

Coding Example

```
SEQUENCE        CONT  A    B                                   COBOL STATEMENT
PAGE SERIAL
1   3 4   6     7     8    12   16   20   24   28   32   36   40   44
002010           ENVIRONMENT DIVISION.
   0 2 0         CONFIGURATION SECTION.
   0 3 0         SOURCE-COMPUTER IBM-S38.
   0 4 0         OBJECT-COMPUTER IBM-S38.
   0 5 0         SPECIAL-NAMES. C01 IS PAGE-TOP.
   0 6 0         INPUT-OUTPUT SECTION.
   0 7 0         FILE-CONTROL.
   0 8 0              SELECT  ------------------>
   0 9 0              ORGANIZATION ------------------>
```

Configuration Section

The Configuration Section describes the computer that compiles the source program and the computer that runs the object program. This section optionally relates IBM-defined function names to user-defined mnemonic-names, specifies the collating sequence to be used, specifies a substitution for the currency sign, and/or interchanges the functions of the comma and the period.

In the Configuration Section, the comma or semicolon can optionally separate successive clauses within a paragraph. In each paragraph, there must be one period; the period must be placed immediately after the last entry in the paragraph.

Format

```
CONFIGURATION SECTION.

SOURCE-COMPUTER. computer-name [ WITH DEBUGGING MODE ].

OBJECT-COMPUTER. computer-name

*****************************************
*  [                                  ]  *
*  [                       { WORDS     }  *
*  [ , MEMORY SIZE integer { CHARACTERS }  *
*  [                       { MODULES   }  *
*  [                                  ]  *
*****************************************

  [ , PROGRAM COLLATING SEQUENCE IS alphabet-name ]

  [ , SEGMENT-LIMIT IS segment-number ].

[ SPECIAL-NAMES. [ function-name IS mnemonic-name ] ...

   [ function-name-2
                                                                          ]
       {IS mnemonic-name, ON STATUS IS condition-name-1 [, OFF STATUS IS condition-name-2]}
       {                                                                  }
       {IS mnemonic-name, OFF STATUS IS condition-name-2 [, ON STATUS IS condition-name-1]}
       {                                                                  } ...
       {ON STATUS IS condition-name-1 [, OFF STATUS IS condition-name-2 ]  }
       {                                                                  }
       {OFF STATUS IS condition-name-2 [, ON STATUS IS condition-name-1 ]  }
                                                                          ]

[                                                                         ]
                 {STANDARD-1                                              }
                 {NATIVE                                                  }
                 {                                                        }
                 {          [ { THROUGH } literal-2                     ] }
                 {literal-1 [ { THRU    }                               ] }
, alphabet-name IS {        [ ALSO literal-3 [, ALSO literal-4 ] ...    ] } ...
                 {                                                        }
                 {                                                        }
                 {          [          [ { THROUGH } literal-6         ] ] }
                 {          [ literal-5 [ { THRU    }                  ] ]...}
                 {          [          [ ALSO literal-7 [, ALSO literal-8 ] ... ] ] }
                 {                                                        }
                                                                          ]

[ , CURRENCY SIGN IS literal-9 ]

[ , DECIMAL-POINT IS COMMA ].
```

SOURCE-COMPUTER Paragraph

The SOURCE-COMPUTER paragraph describes the computer that compiles the source program. The computer name should be coded as: IBM-S38.

With the exception of the WITH DEBUGGING MODE clause, the SOURCE-COMPUTER paragraph is syntax-checked, but is treated as documentation. The WITH DEBUGGING MODE clause is described under “DEBUGGING FEATURES” on page 517.

OBJECT-COMPUTER Paragraph

The OBJECT-COMPUTER paragraph identifies the computer that runs the object program. Computer-name must be the first entry in the OBJECT-COMPUTER paragraph. The other clauses can be specified in any order. The computer-name should be coded as: IBM-S38.

MEMORY SIZE Clause

The MEMORY SIZE clause is syntax-checked, but is treated as documentation.

PROGRAM COLLATING SEQUENCE Clause

The PROGRAM COLLATING SEQUENCE clause specifies the collating sequence used in a program. The collating sequence associated with the specified alphabet-name must be defined in the SPECIAL-NAMES paragraph. The program collating sequence applies to the following nonnumeric comparisons:

- Those comparisons explicitly specified in IF, PERFORM, and SEARCH statements
- Those comparisons implicitly specified in STRING, INSPECT, and UNSTRING statements
- Those comparisons implicitly specified in MERGE or SORT statements that do not specify a COLLATING SEQUENCE phrase.

When the PROGRAM COLLATING SEQUENCE clause is omitted, the EBCDIC collating sequence is used. See Appendix F, “EBCDIC and ASCII Collating Sequences” for the complete EBCDIC collating sequence.

SEGMENT-LIMIT Clause

The SEGMENT-LIMIT clause is described under “SEGMENTATION FEATURE” on page 503.

SPECIAL-NAMES Paragraph

The SPECIAL-NAMES paragraph relates IBM-specified function-names to user-specified mnemonic-names. This paragraph specifies a collating sequence that is associated with an alphabet-name, a substitute character for the currency sign, and the interchange of the comma and decimal point in PICTURE clauses and numeric literals. The clauses can be specified in any order.

Function-Name-1 Clause

Note: Card devices are not supported by System/38-Compatible COBOL, even though the devices are accepted by the syntax checker.

Function-name-1 specifies system devices or standard system actions taken by the compiler.

The associated mnemonic-name is required. The mnemonic-name is formed according to the rules for a user-defined word and must contain at least one alphabetic character.

Note: The SEU Syntax Checker requires that the first clause of the following paragraphs be entered on the same line as the paragraph name.

- SPECIAL-NAMES
- PROGRAM-ID
- AUTHOR
- INSTALLATION
- DATE-WRITTEN
- DATE-COMPILED
- SECURITY
- SOURCE-COMPUTER
- OBJECT-COMPUTER.

This is not a requirement for compiling.

Table 6 shows the actions that are associated with mnemonic names for function-name-1. Each of these functions can appear only once in the SPECIAL-NAMES paragraph.

Table 6. Choices of Function-Name-1 and Action Taken

Function-name-1	Statement where mnemonic-name associated with function-name is used	Usage
CSP	WRITE	Suppress spacing when printing a line. Use only when PRINTER is the device. See "FILE-CONTROL Paragraph" later in this chapter.
C01	WRITE	Skip to the next page. Use only when PRINTER is the device. See "FILE-CONTROL Paragraph" later in this chapter.
S01, S02, S03, S04, S05	WRITE	Select stackers on a card punch file. S01 through S04 select stackers 1 through 4, and S05 selects stacker 1 on the IBM 5424. Use only when PUNCH, PUNCHPRINT, or PRINT is the device. See "FILE-CONTROL Paragraph" later in this chapter.
ATTRIBUTE-DATA	ACCEPT	Retrieve attribute data about a program device acquired by a TRANSACTION file, but only when the file is open. See "ACCEPT Statement" in Chapter 5, "Interactive Processing Considerations and Example Programs."
I-O-FEEDBACK	ACCEPT	Give information about the last I-O operation on a file, but only when the file is open. See "ACCEPT Statement" in Chapter 10, "Procedure Division."
OPEN-FEEDBACK	ACCEPT	Give information about a file, but only when the file is open. See "ACCEPT Statement" in Chapter 10, "Procedure Division."
CONSOLE, SYSTEM-CONSOLE	ACCEPT,DISPLAY	Communicate with the system operator's message queue (QSYSOPR).
LOCAL-DATA	ACCEPT,DISPLAY	Retrieve data from, or moves data to the local data area created by the system for every job. See "ACCEPT Statement" and "DISPLAY Statement" in Chapter 10, "Procedure Division."
REQUESTOR	ACCEPT,DISPLAY	Communicate with the user work station (interactive jobs) or the batch input stream or job log (batch jobs).

Function-Name-2 Clause

Function-name-2 can be defined as UPSI-0 through UPSI-7 or as SYSTEM-SHUTDOWN.

User Program Status Indicator (UPSI): Function-name-2 can define eight 1-byte program switches, UPSI-0 through UPSI-7.

Each UPSI is a User Program Status Indicator switch. At least one condition-name must be associated with each UPSI switch specified. UPSI-0 through UPSI-7 are COBOL names that identify program switches defined outside the COBOL program

at object time. Their contents are considered to be alphanumeric. A value of zero is off; a value of one is on.

Each switch represents one byte from the eight-character SWS parameter of the control language CHGJOB, SBMJOB, JOB, and JOBD commands as follows:

```
UPSI-0  First byte (leftmost)
UPSI-1  Second byte
UPSI-2  Third byte
.           .
.           .
.           .
UPSI-7  Eighth byte (rightmost)
```

One condition-name must be associated with each function-name-2; a second condition-name is optional. One condition-name can be associated with the ON status; another can be associated with the OFF status. Establishing condition-names for the ON or OFF status of a switch permits testing the setting of that switch.

Each condition-name is formed according to the rules for a user-defined word, and the condition-name must contain at least one alphabetic character.

In the Procedure Division, the UPSI switch status is tested through the associated condition-name(s). Each condition-name is the equivalent of a level-88 item. The associated mnemonic-name, if specified, is considered the conditional variable and can be used for qualification.

Programming Notes: UPSI switches are useful for processing special conditions within a program, such as year-beginning or year-ending processing. At the beginning of the Procedure Division, an UPSI switch can be tested; if it is ON, the special branch is taken.

SYSTEM-SHUTDOWN: SYSTEM-SHUTDOWN is an internal switch that is set to ON status when the system operator causes the system to be in a shutdown-pending state or when the job is being canceled in a controlled manner. The associated ON or OFF condition-names can be referenced anywhere a condition-name is valid. Their status cannot be altered by the program.

Coding Example

This coding example assigns mnemonic-names to some commonly used function-names in the SPECIAL-NAMES paragraph.

```
SPECIAL-NAMES.  SYSTEM-CONSOLE IS SYSTM,
    REQUESTOR IS WORK-STATION,
    C01 IS NEXT-PAGE,
    LOCAL-DATA IS LOCAL-DATA-AREA,
    ATTRIBUTE-DATA IS ATTRB-DATA,
    SYSTEM-SHUTDOWN IS SHUTDOWN-SWITCH,
        ON STATUS IS SHUTDOWN-PENDING,
    UPSI-0 IS UPSI-SWITCH-0,
        ON STATUS IS U0-ON,
        OFF STATUS IS U0-OFF,
    UPSI-1 IS UPSI-SWITCH-1,
        ON STATUS IS U1-ON,
        OFF STATUS IS U1-OFF,
    IBM-ASCII IS STANDARD-1,
    CURRENCY-SIGN IS "Y".
```

Alphabet-Name Clause

The alphabet-name clause provides a means of relating an alphabet-name to a specified character code set or collating sequence.

The alphabet-name specifies a collating sequence in one of the following:

- The PROGRAM COLLATING SEQUENCE clause in the OBJECT-COMPUTER paragraph
- The COLLATING SEQUENCE phrase of the SORT or MERGE statement.

The EBCDIC collating sequence is used when NATIVE is specified or when the alphabet-name clause is omitted.

The ASCII (American National Standard Code for Information Interchange) collating sequence is used when STANDARD-1 is specified.

Literal Phrase: The literal phrase of the alphabet-name clause processes internal data in collating sequences other than NATIVE or STANDARD-1.

When the literal phrase is specified, the collating sequence to be used is specified by the user according to the following rules:

- The order in which literals appear specifies the ordinal number, in ascending sequence, of the character(s) in this collating sequence.
- Each numeric literal specified must be an unsigned integer and must have a value from 1 through 256 (the maximum number of characters in the EBCDIC character set). The value of each literal specifies the relative position of a character within the EBCDIC character set. For example, the literal 112 represents the EBCDIC character ?, the literal 234 represents the EBCDIC character Z, the literal 241 represents the EBCDIC numeric character 0.
- Each character in a nonnumeric literal represents that character in the EBCDIC set. If the nonnumeric literal contains more than one character, each character, starting with the leftmost, is assigned a successively ascending position within this collating sequence.
- Any EBCDIC characters not explicitly specified assume positions in this collating sequence higher than any of the explicitly specified characters. The relative order of the unspecified characters within the EBCDIC set remains unchanged.
- Within one alphabet-name clause, a given character must not be specified more than once.

- Each nonnumeric literal associated with a THROUGH or ALSO phrase must be one character in length.
- When the THROUGH phrase is specified, the contiguous EBCDIC characters beginning with the character specified by literal-1 and ending with the character specified by literal-2 are assigned successively ascending positions in this collating sequence. This sequence may be either ascending or descending within the original EBCDIC sequence. For example, if the characters Z through S are specified, then for this collating sequence the ascending values are: ZYXWVUTS
- When the ALSO phrase is specified, the EBCDIC characters specified as literal-1, literal-3, literal-4, and so on are assigned to the same position in this collating sequence. For example, if **"D"** ALSO **"N"** ALSO 112 ALSO **"%"** is specified, then for this collating sequence the characters D, N, ?, and % are all considered to be in the same position in the collating sequence.

If specified as literals in the SPECIAL-NAMES paragraph, the figurative constants HIGH-VALUE and LOW-VALUE are associated with hex 00 and hex FF respectively.

After all clauses in the SPECIAL-NAMES paragraph are processed, the character having the highest ordinal position in this collating sequence is associated with the figurative constant HIGH-VALUE. If more than one character has the highest position because the ALSO phrase is specified, the last character specified (or defaulted to when any characters within the native collating sequence are not explicitly specified) is considered to be the HIGH-VALUE character for procedural statements such as DISPLAY, or as the sending field in a MOVE statement. If all characters within the native collating sequence were explicitly specified and the ALSO phrase example given above was specified as the high-order characters of this collating sequence, the HIGH-VALUE character would be %.

After all clauses in the SPECIAL-NAMES paragraph are processed, the character having the lowest ordinal position in this collating sequence is associated with the figurative constant LOW-VALUE. If more than one character has the lowest position because the ALSO phrase is specified, the first character specified is the LOW-VALUE character. If the ALSO phrase example given above were specified as the low-order characters of the collating sequence, then the LOW-VALUE character would be D.

Alphabet-Name Clause Examples: The following examples illustrate some uses for the alphabet-name clause.

If PROGRAM COLLATING SEQUENCE IS USER-SEQUENCE; if the alphabet-name clause is specified as USER-SEQUENCE IS **"D", "E", "F"**; and if two Data Division items are defined as follows:

```
01 ITEM-1 PIC X(3) VALUE "ABC".
01 ITEM-2 PIC X(3) VALUE "DEF".
```

then the comparison IF ITEM-1 > ITEM-2 is true.

Characters D, E, and F are in ordinal positions 1, 2, and 3 of this collating sequence. Characters A, B, and C are in ordinal positions 197, 198, and 199 of this collating sequence.

If the alphabet-name clause is USER-SEQUENCE IS 1 THRU 247, 251 THRU 256, "7", ALSO "8", ALSO "9"; if all 256 EBCDIC characters have been specified; and if the two Data Division items are specified as follows:

```
01 ITEM-1 PIC X(3) VALUE HIGH-VALUE.
01 ITEM-2 PIC X(3) VALUE "787".
```

then both of the following comparisons are true:

```
IF ITEM-1 = ITEM-2 . . .
IF ITEM-2 = HIGH-VALUE . . .
```

They compare as true because the values "7", "8", and "9" all occupy the same position (HIGH-VALUE) in this USER-SEQUENCE collating sequence.

If the alphabet-name clause is specified as USER-SEQUENCE IS **"E", "D", "F"** and a table in the Data Division is defined as follows:

```
05 TABLE A OCCURS 6 ASCENDING KEY IS
   KEY-A INDEXED BY INX-A.
   10 FIELD-A ...
   10 KEY-A ...
```

and if the contents in ascending sequence of each occurrence of KEY-A are A, B, C, D, E, G, then the results of processing a SEARCH ALL statement for this table will be invalid because the contents of KEY-A are not in ascending order. The proper ascending order would be E, D, A, B, C, G.

CURRENCY SIGN Clause

The literal that appears in the CURRENCY SIGN clause defines the currency symbol to be used in the PICTURE clause. The literal must be a one-character nonnumeric literal and must not be any of the following characters:

- Digits 0 through 9
- Alphabetic characters A B C D L P R S V X Z or the space
- Special characters . (+ *) ; - / , = "

When the CURRENCY SIGN clause is omitted, only the dollar sign ($) may be used as the PICTURE symbol for the currency sign.

DECIMAL-POINT IS COMMA Clause

When the DECIMAL-POINT IS COMMA clause is specified, the functions of the period and the comma are exchanged in PICTURE character-strings and in numeric literals.

Input-Output Section

The Input-Output Section defines each file, identifies its external storage medium, assigns the file to one or more input/output devices, and also specifies information needed for efficient transmission of data between the external medium and the COBOL program.

Files

COBOL supports three categories of files: data base files, device files and DDM files.

Data Base Files

Data base files allow information to be permanently stored on the system. Multiple programs can access this information in different ways.

A data base file is subdivided into groups of records called members. Every file has at least one member.

There are two types of data base files: physical files and logical files.

Physical Files: A physical file is a file that actually contains data records. This makes physical files similar to disk files on other systems. A physical file can contain only fixed-length records, all of which have the same format.

Logical Files: A logical file is a data base file through which data from one or more physical files can be accessed. The format and organization of this data is different from that of the data in the physical file(s). Each logical file can define a different access path (index) for the data in the physical file(s). Each logical file can exclude and reorder the fields defined in the physical file(s).

Device Files

A device file reads from or writes to a device attached to the system. A device file controls the transfer of data between the physical device or a remote system, and the program.

This manual uses the term *file* to refer to any of these device types.

DDM Files

Distributed Data Management (DDM) allows you to access data that resides on remote systems that support DDM. DDM files are supported by the COBOL compiler. You can retrieve, add, update or delete data records in a file that resides on another system.

When you compile a System/38-Compatible COBOL source program that is on a remote system, the compiler expects a source type of CBL38. If the source type is not CBL38, the compiler issues a message indicating that it encountered an unexpected source member type. To resolve this discrepancy, you should recompile the program in the environment indicated by the source member type, or change the source member type, or use the correct compiler indicated by the source member type.

For more information about accessing remote files, refer to the *DDM Guide*.

Paragraphs

The Input-Output Section is divided into two paragraphs: the `FILE-CONTROL` paragraph, which names and associates the files with the external media, and the `I-O-CONTROL` paragraph, which defines special input/output techniques to be used.

Format

```
┌ INPUT-OUTPUT SECTION.
  FILE-CONTROL. { file-control-entry } . . .

[ I-O-CONTROL. input-output-control-entry ] ┐ .
```

The exact contents of the Input-Output Section depend on the file organization and access methods used to process the file. The following summary gives some background for the file processing techniques available in System/38-Compatible COBOL.

File Processing Summary

The method used to process a file in a COBOL program depends on the data organization of the file and on the access mode used.

Appendix E, “File Structure Support Summary and Status Key Values” on page 543 summarizes which clauses and statements are required and which clauses and statements are optional for each access mode and device.

The following paragraphs describe both the types of data organization, and the access modes available in COBOL. See Chapter 7, “System/38-Compatible COBOL Programming Considerations” for information about COBOL file processing in relation to AS/400 file processing.

Data Organization

In a COBOL program, data organization can be sequential, indexed, relative, or TRANSACTION.

Records can be fixed or variable in length. For all files other than tape, variable length records are stored as fixed length records of the maximum size specified for the file.

Sequential Organization: With this organization, records are placed in the file consecutively, without keys, in the order they are written (arrival sequence). Once established, this relationship does not change, with the exception that a file can be extended. Both data base files and device files can have sequential organization.

Indexed Organization: With this organization, each record in the file has one embedded key that is associated with an index. The index provides a logical path to the data records according to the contents of the associated embedded record key data item (key sequence).

When records are inserted, updated, or deleted, they are identified solely by the value of their record key. Thus, the value in each record key data item must be unique and must not be changed when the record is updated. The key used for any specific input/output request is known as the *key of reference.*

Only data base files can have indexed organization.

IBM Extension

A logical file that is opened for OUTPUT does not remove all records in the physical file on which it is based. Instead, the file is opened to allow only write operations, and the records are added to the file.

End of IBM Extension

Relative Organization: With this organization, each record in the file is identified by its relative record number. The file can be thought of as a serial string of areas, each of which can contain one record. Each of these areas is identified by a relative record number; record storage and retrieval are based on this number. For example, the first record area is addressed by relative record number 1, and the tenth is addressed by relative record number 10, whether or not records have been written in the second through ninth record areas. Relative files must be assigned to DISK or DATABASE.

New relative files opened for OUTPUT are initialized with all records deleted. In the absence of command language override, the number of records in a newly created file is the number of records specified at file creation time including all increments. Any attempt to extend a relative file beyond its current size results in a boundary violation.

Relative record number processing can be used for a physical file or for a logical file that is based on only one physical file.

IBM Extension

TRANSACTION Organization: Work station and data communication files can have TRANSACTION organization. See Chapter 5, “Interactive Processing Considerations and Example Programs” for a discussion of this organization.

End of IBM Extension

Access Modes

Access mode is a COBOL term that defines the manner in which data in a logical or physical file is to be processed. The three access modes are sequential, random, and dynamic.

Sequential Access Mode: This access method allows records of a file to be read and written in a serial manner. The order of reference is implicitly determined by the position of a record in the file.

Random Access Mode: This access method allows records to be read and written in a user-specified manner. The control of successive references to the file is expressed by specifically defined keys supplied by the user.

Dynamic Access Mode: This access method allows a specific input/output request to determine the access mode. Thus records can be processed sequentially and/or randomly.

Access Mode Allowed for Each File Type

Sequential Files: Files with sequential organization can be accessed only sequentially. The sequence in which records are accessed is the order in which the records were originally written (arrival sequence).

Indexed Files: All three access modes are allowed.

In the sequential access mode, the sequence in which records are accessed is determined by the `RECORD KEY` value.

In the random access mode, the sequence in which records are accessed is controlled by the user. The desired record is accessed by placing the value of its record key in the `RECORD KEY` data item defined for that file.

In the dynamic access mode, the user can change from sequential access to random access by using appropriate input/output statements.

Relative Files: All three access modes are allowed.

In the sequential access mode, the sequence in which records are accessed is the ascending order of the relative record numbers of all records that currently exist within the file.

In the random access mode, the sequence in which records are accessed is controlled by the user. The desired record is accessed by placing its relative record number in a `RELATIVE KEY` data item.

In the dynamic access mode, the user can change from sequential access to random access by using appropriate input/output statements.

TRANSACTION Files: See Chapter 5, “Interactive Processing Considerations and Example Programs” for a discussion of access mode considerations for `TRANSACTION` files.

FILE-CONTROL Paragraph

The `FILE-CONTROL` paragraph contains one or more file-control entries. A file-control entry associates a file in the COBOL program with an external medium, and this entry allows specification of file organization, access mode, and other information. The format of a file-control entry varies with the type of file described. The formats for the `FILE-CONTROL` paragraph are as follows:

Note: Card devices are not supported by System/38-Compatible COBOL, even though the card devices and related language elements are accepted by the syntax checker.

Format 1–Sequential File Entries[1]

```
SELECT [ OPTIONAL ] file-name
                                  ******************************
      ASSIGN TO assignment-name-1 * [ , assignment-name-2 ] ... *
                                  ******************************

  ***********************************
  * [ RESERVE integer-1 [ AREA  ] ] *
  *                     [ AREAS ]   *
  ***********************************

     [ ORGANIZATION IS SEQUENTIAL ]

     [ ACCESS MODE IS SEQUENTIAL ]

     [ FILE STATUS IS data-name-1 ] .
```

[1] Format 1 of the file-control entry is used with (READER, PUNCH, PUNCHPRINT, PRINT, PRINTER, TAPEFILE, DISKETTE, FORMATFILE, DISK, and DATABASE) files.

Format 2–Indexed File Entries (DISK, DATABASE)

```
SELECT file-name
                                  ******************************
     ASSIGN TO assignment-name-1 * [ , assignment-name-2 ] ... *
                                  ******************************

  ***********************************
  * [ RESERVE integer-1 [ AREA  ] ] *
  *                     [ AREAS ]   *
  ***********************************

     ORGANIZATION IS INDEXED

   [                  { SEQUENTIAL } ]
   [ ACCESS MODE IS { RANDOM     }   ]
   [                  { DYNAMIC    }   ]

     RECORD KEY IS { EXTERNALLY-DESCRIBED-KEY } [ WITH DUPLICATES ]
                   { data-name-2              }

   [ FILE STATUS IS data-name-1 ] .
```

Format 3–Relative File Entries (DISK, DATABASE)

```
SELECT file-name
                                 ******************************
     ASSIGN TO assignment-name-1 * [ , assignment-name-2 ] ... *
                                 ******************************

  ***********************************
  *  [ RESERVE integer-1 [ AREA  ] ]  *
  *  [                   [ AREAS ] ]  *
  *                                   *
  ***********************************

     ORGANIZATION IS RELATIVE

   [                 { SEQUENTIAL [ , RELATIVE KEY IS data-name-3 ] }  ]
   [                 {                                              }  ]
   [ ACCESS MODE IS  { { RANDOM  } , RELATIVE KEY IS data-name-3    }  ]
   [                 { { DYNAMIC }                                  }  ]


   [ FILE STATUS IS data-name-1 ] .
```

Format 4–Sort or Merge File Entries

```
                 ************************************************************
SELECT file-name * ASSIGN TO assignment-name-1 [ , assignment-name-2 ] ... * .
                 ************************************************************
```

Format 5–TRANSACTION File Entries (WORKSTATION)

```
SELECT file-name
                                 ******************************
     ASSIGN TO assignment-name-1 * [ , assignment-name-2 ] ... *
                                 ******************************

     ORGANIZATION IS TRANSACTION

   [ ACCESS MODE IS { SEQUENTIAL                            }  ]
   [                { DYNAMIC, RELATIVE KEY IS data-name-3 }  ]


   [ FILE STATUS IS data-name-1 { , data-name-5 } ]


   [ CONTROL-AREA is data-name-6 ] .
```

See “File-Control Entry” on page 122 for a discussion of Format 5.

FILE-CONTROL Paragraph–General Considerations

Each file described in an FD or SD entry in the Data Division must be described in only one entry in the FILE-CONTROL paragraph. Each file specified in a file-control entry must have a file description in the Data Division.

The keyword FILE-CONTROL can appear only once, at the beginning of the FILE-CONTROL paragraph. The word FILE-CONTROL must begin in Area A, and it must be followed by a period and a space.

Each file-control entry must begin in Area B with a SELECT clause. The order in which other clauses appear is not significant.

Each clause within a file-control entry can optionally be separated from the next by a comma or semicolon followed by a space. Each file-control entry ends with a period and a space.

Each data-name must appear in a Data Division data description entry. Each data-name can be qualified but cannot be subscripted or indexed.

SELECT Clause

Each file-name specified in a SELECT clause must have an FD or SD entry in the Data Division. A file-name must conform to the rules for a COBOL user-defined name, must contain at least one alphabetic character, and must be unique within this program.

Sequential File Considerations: The OPTIONAL phrase can be specified only for input files with sequential organization. It must be specified for input files that are not necessarily present each time the program is run.

ASSIGN Clause

The ASSIGN clause associates a file with an external medium. The assignment-name makes the association between the file and the external medium. For sort or merge files (associated with an SD entry), no external medium is used. The related ASSIGN clause is only validity checked. It is not actually used for I-O.

Assignment-name consists of three parts:

- Device
- file name
- Attribute.

It has the following general structure:

```
Device [–file name [–attribute] ]
```

Device: This part of assignment-name specifies the type of device that the file will use. The compiler can then check whether the file is described and used in a consistent manner. See “Device Independence/Device Dependence” on page 206 for further information.

The compiler does not check whether the device associated with the external file is of the type specified in the device portion of assignment-name. For example, assignment-name could be TAPEFILE-ABCD and ABCD could be created with a Create diskette (CRTDKTF) CL command. The compiler would provide no diagnostics unless the I-O verbs were used in an inconsistent manner for TAPEFILE. At run time,

OS/400 could either issue an escape message or ignore the function if it was not applicable to the device. See the *System/38 CPF Programmer's Guide* for further information on overriding files in relation to the System/38 environment.

IBM Extension

The device that the file will use can be changed at run time with the OVRxxxF CL command. To ensure consistent results, the device associated with the file should correspond to that given in the assignment-name.

End of IBM Extension

Device can be any of the following:

Note: Card devices are not supported by System/38-Compatible COBOL, even though the devices are accepted by the syntax checker.

Device	Associated File
READER	Card file
PUNCH	Card file
PUNCHPRINT	Card file
PRINT	Card file
PRINTER[5]	Printer file
FORMATFILE[6]	Printer file
TAPEFILE	Tape file
DISKETTE	Diskette file
DISK[7]	Any physical data base file or single format logical data base file
DATABASE[8]	Any data base file (including DDM file)
WORKSTATION	Display file, communications file, binary synchronous communications file, or mixed file.

For more information on how to use externally described printer files see “FORMATFILE Files” on page 231.

Note: See “DISK and DATABASE File Considerations” on page 236 for further information.

File Name: This part of assignment-name must be an unhyphenated, one- through ten-character system name of the actual external file (physical or logical data base, or device). This external file has to be created before compiling the program only when it is used by a Format 2 COPY statement within this program.

5 PRINTER should be specified for program described printer files only.

6 FORMATFILE should be specified for externally described printer files only.

7 When DISK is the device, data base extensions cannot be used.

8 When DATABASE is the device, externally described data and data base extensions can be used.

For data base files, the member name cannot be specified in the program. If a member other than the first member is to be specified, the Override with Data Base File (OVRDBF) CL command must be used at run time to specify the member name.

This file name is the name of the OS/400 object that is displayed by the Display Program References (DSPPGMREF) command. Since no external medium is used for an SD file, the DSPPGMREF command does not list any files defined for an SD file.

The file name can be changed at run time with the TOFILE parameter of the OVRxxxF CL command. To ensure consistent results, the device type associated with the TOFILE parameter should be the same as that specified in the assignment-name.

Attribute: This part of assignment-name can be one of the following:

- — hopper [– association]
- — SI.

Hopper must be either P or S to specify the primary or secondary hopper for card device files. If neither P nor S is specified for a card device file, the HOPPER parameter on the Create Card File (CRTCRDF) or Change Card File (CHGCRDF) CL commands is used.

Note: These commands are only supported on System/38.

Association must be any single-digit number from 0 through 9. It can be used only if the primary (P) hopper is specified for the file. All unit record card files that have the same association number are assigned to the same unit record card device, and must use the same external file name (see Appendix B, “Associated Card File Processing”).

SI indicates that a separate indicator area has been specified in the DDS for a FORMATFILE or WORKSTATION file. See “Indicators” on page 92 for more information on the use of the SI attribute.

See “Device Independence/Device Dependence” on page 206 for further information on the ASSIGN clause.

The valid entries for each field of the assignment-name vary with the device. The valid combinations of fields are shown in Figure 68 on page 287.

In formats 1, 2, and 3, the second and subsequent assignment-names are syntax-checked, but are treated as documentation. In format 4, the entire ASSIGN clause is syntax-checked, but is treated as documentation.

RESERVE Clause

The RESERVE clause is syntax-checked, but is treated as documentation.

ORGANIZATION Clause

The ORGANIZATION clause specifies the logical structure of the file. The file organization is established at the time the file is created and cannot subsequently be changed. When the ORGANIZATION clause is omitted, ORGANIZATION IS SEQUENTIAL is assumed.

IBM Extension

For data base files, the ORGANIZATION clause indicates the current program usage of the file in the program. Therefore, the same data base file can use SEQUENTIAL, INDEXED (assuming a keyed sequence access path exists), or RELATIVE in the ORGANIZATION clause. This is true regardless of what is specified in other programs that use this file.

End of IBM Extension

Notes:

1. A keyed sequence access path is always created when a key is specified in the DDS that was used as input to the Create Physical File (CRTPF) or the Create Logical File (CRTLF) CL command.
2. Card devices are not supported by the System/38-Compatible COBOL even though the devices are accepted by the syntax checker.

Device	AS/400 File Name	Default system. File Name	Hopper	Associ-ation	SI
READER	O	QCARD96	O	O	N
PUNCH	O	QCARD96	O	O	N
PUNCHPRINT	O	QCARD96	O	O	N
PRINT	O	QCARD96	O	O	N
PRINTER	O	QPRINT	N	N	N
FORMATFILE	R		N	N	O
TAPEFILE	O	QTAPE	N	N	N
DISKETTE	O	QDKT	N	N	N
DISK	R		N	N	N
DATABASE	R		N	N	N
WORKSTATION	R		N	N	O

R=Required
O=Optional
N=Not Allowed

Figure 68. Valid Entries for the Assignment-Name

Sequential File Considerations: When ORGANIZATION IS SEQUENTIAL is specified or implied, a predecessor-successor relationship of the records in the files is established by the order in which records are placed in the file when it is created or extended (arrival sequence access path).

Indexed File Considerations: When ORGANIZATION IS INDEXED is specified, the position of each logical record in the file is determined by the key sequence access path created with the file and maintained by the system. The access path is based on an embedded key within the file's records.

Relative File Considerations: When ORGANIZATION IS RELATIVE is specified, the position of each record in the file is determined by its relative record number within the arrival sequence access path.

ACCESS MODE Clause

The ACCESS MODE clause defines the manner in which the records of the file are made available for processing. When this clause is omitted, ACCESS IS SEQUENTIAL is assumed.

Sequential File Considerations: For files with sequential organization, records in the file are accessed in the order they are written when the file is created or extended (arrival sequence). Whether ACCESS IS SEQUENTIAL is specified or omitted, sequential access is always assumed.

Indexed File Considerations: For files with indexed organization, the access mode can be SEQUENTIAL, RANDOM, or DYNAMIC.

When ACCESS IS SEQUENTIAL is specified or implied, records in the file are accessed in the sequence of ascending record key values within the index.

IBM Extension

When using an externally described file, if the DDS keyword DESCEND is used when the field is specified as a key field, the records in the file are accessed in the sequence of descending record key values within the index.

End of IBM Extension

When ACCESS IS RANDOM is specified, the value placed in the RECORD KEY data item specifies the record to be accessed.

When ACCESS IS DYNAMIC is specified, records in the file can be accessed sequentially or randomly, depending on the form of the specific input/output request.

Relative File Considerations: For files with relative organization, the access mode can be SEQUENTIAL, RANDOM, or DYNAMIC.

When ACCESS IS SEQUENTIAL is specified or implied, records in the file are accessed in the ascending sequence of relative record numbers in the arrival sequence access path.

When ACCESS IS RANDOM is specified, the value placed in the RELATIVE KEY data item specifies the record to be accessed.

When ACCESS IS DYNAMIC is specified, records in the file can be accessed sequentially or randomly, depending on the form of the specific input/output request.

RELATIVE KEY Phrase: The RELATIVE KEY phrase specifies the relative record number for a specific record in a relative file.

Data-name-3 is the RELATIVE KEY data item. It must be defined as an unsigned integer data item and must not be defined in a record description entry associated with this relative file. That is, the RELATIVE KEY is not part of the record.

When ACCESS IS SEQUENTIAL is specified, the RELATIVE KEY phrase need not be specified unless the START statement is used. When the START statement is used, the system uses the contents of the RELATIVE KEY data item to determine the record at which sequential processing is to begin.

If a value is placed in the RELATIVE KEY data item and a START statement is not used, the value is ignored and processing begins with the first record in the file.

IBM Extension

When the file is opened, the POSITION parameter on the OVRDBF CL command can be used to set the current record pointer. This causes processing to begin with a record other than the first record. See the *CL Reference* for further information.

End of IBM Extension

When ACCESS IS RANDOM or ACCESS IS DYNAMIC is specified, the RELATIVE KEY phrase must be specified. For each random processing request, the contents of the RELATIVE KEY data item are used to communicate a relative record number to the system.

TRANSACTION File Considerations: See Chapter 5, “Interactive Processing Considerations and Example Programs.”

RECORD KEY Clause (Indexed File)

The RECORD KEY clause must be specified for an indexed file. The RECORD KEY clause specifies the data item within the record that is the record key for an indexed file. The values contained in the record key data item must be unique among records in the file.

IBM Extension

The DUPLICATES phrase can only be specified for files assigned to DATABASE. This allows the file to have keys with the same values. If the file has multiple formats, two keys in different formats have the same values only when the key lengths and the contents of the keys are the same.

For example, given a file with the following two formats:

Format F1 with keys A, B, C
Format F2 with keys A, B, D.

If fields C and D are the same length, have the same data type, and have the same values, the file would contain two records with a duplicate key. The term *duplicate key* applies only to a complete record key for the format. A record key for the format consists of the key field(s) defined for a DDS format for records residing on the data base. The term does not apply to the common key for the file (only fields A and B in the above example).

Users can indicate DUPLICATES on the RECORD KEY clause. A file status of 95 is returned after a successful open when:

- The DUPLICATES phrase is specified in the COBOL program and the file was created with UNIQUE specified in DDS.

- The DUPLICATES phrase is not specified in the COBOL program and the file was created allowing nonunique keys.

Processing files when either of these conditions exist can cause unpredictable results.

To ensure that the proper duplicate record is updated or deleted in a file that allows duplicates and is processed randomly, the last input/output statement processed prior to the processing of the REWRITE or DELETE statement must be a successfully processed READ statement for the record to be deleted or rewritten.

If the DDS file level keyword LIFO (last-in-first-out) is specified, the duplicate records within a physical file are retrieved in a last-in-first-out order.

End of IBM Extension

Data-name-2 is the RECORD KEY data item. It must be described as a fixed-length alphanumeric item within a record description entry associated with the file. The length of the record key is restricted; the key length, in characters, plus the number of fields cannot exceed 120. See the *DDS Reference* for more information.

IBM Extension

The RECORD KEY data item, data-name-2, can be a numeric item when the file is assigned to a DATABASE device type. The numeric item can have a usage of DISPLAY, COMP (COMP-3), or COMP-4.

Depending on the keywords specified for the data item in DDS, the keyed sequence access path can be by algebraic value. See the ABSVAL, DIGIT, SIGNED, and ZONE keywords in the *DDS Reference*. If one of these keywords is specified, its name appears in a comment table in the COBOL source listing under the heading TYPE. If no keyword is specified, the table entry is the data type specified in DDS. The table entry AN indicates that the data type is alphanumeric (specified in DDS as A). The table entry N indicates that the data type is numeric (specified in DDS as P, S, or B).

The keywords specified for the data item in DDS can modify record sequence. See the ALTSEQ, DIGIT, and ZONE keywords in the *DDS Reference*. If none of these keywords are specified, the records are ordered according to the EBCDIC collating sequence.

End of IBM Extension

The data description of data-name-2 and its relative location within the record must be the same as the ones used when the file was defined in DDS.

The record description that defines data-name-2 will always be used to access the record key field for the I-O operation.

IBM Extension

The reserved word EXTERNALLY-DESCRIBED-KEY can specify that the key(s) for this file are those that are externally described in DDS. The keys are determined by the

record formats that are copied by the COPY statement, DDS or DD format, under the FD for this file.

The key can start at different offsets within the buffer for each format. In this situation, care must be used when changing from one record format to another, using a random READ or START statement. The key must be placed in the record format at the correct offset in the format that will be used in the random access of the file. Unpredictable results can occur if the key for the desired record is based on data that was part of the last record read. This is because the movement of the data to the key field can involve overlapping fields.

The key within a format can be made up of multiple, noncontiguous (not adjacent) fields. When using EXTERNALLY-DESCRIBED-KEY for a logical file, the key fields defined for a record format in DDS must also be fields defined in that format. Therefore, fields renamed in DDS, or fields that are part of concatenated fields in DDS cannot be used as keys. Only those record formats copied in within the FD for the file should be referenced by the FORMAT phrase. If a format is referenced that is defined within the file, but that format has not been copied into the program, the key is built using the key field(s) defined for the first record format that was copied. This can cause unpredictable results.

End of IBM Extension

FILE STATUS Clause

The FILE STATUS clause allows the user to monitor the processing of each input/output request for the file.

Data-name-1 is the status key data item. Data-name-1 must be defined in the Data Division as a two-character alphanumeric item and must not be defined in the File Section.

When the FILE STATUS clause is specified, the system moves a value into the status key data item after each input/output request that explicitly or implicitly refers to this file. The value indicates the run status of the statement. When the compiler generates code to block output records or unblock input records, file status values that are caused by OS/400 exceptions are set only when a block is processed. See Appendix E, “File Structure Support Summary and Status Key Values” for a description of the possible values. See Chapter 7, “System/38-Compatible COBOL Programming Considerations” for more information on blocking output records and unblocking input records.

IBM Extension

An extended file status data item may be specified for TRANSACTION file processing. See Chapter 5, “Interactive Processing Considerations and Example Programs” for more information.

End of IBM Extension

I-O-CONTROL Paragraph

The I-O-CONTROL paragraph specifies when checkpoints are to be taken and what storage areas are to be shared by different files and optimization techniques. The I-O-CONTROL paragraph is optional in a COBOL program.

Format

```
[ I-O-CONTROL.

    ******************************************************
    * [ RERUN ON assignment-name                         *
    *                                                    *
    *   EVERY integer-1 RECORDS OF file-name-1 ] . . . *
    ******************************************************
    [      [          ]                                                  ]
    | SAME | RECORD   |  AREA FOR file-name-2 { , file-name-3 } . . .  | . . .
    [      [ SORT     ]                                                  ]
           [ SORT-MERGE ]
    ************************************************************
    * [                                                        *
    * | MULTIPLE FILE TAPE CONTAINS                            *
    * [                                                        *
    *   file-name-4 [ POSITION integer-2 ]                     *
    *                                                          *
    *                                                          *
    *   [ file-name-5 [ POSITION integer-3 ] ] . . . ] . . .   *
    *                                                          *
    ************************************************************
    ╔══════════════════════════════════════╗
    ╟ [ COMMITMENT CONTROL FOR file-name-6  ╢
    ╟                                      ╢
    ╟ [ , file-name-7 ]  . . . ]           ╢   . ]
    ╚══════════════════════════════════════╝
```

The keyword I-O-CONTROL can appear only once, at the beginning of the I-O-CONTROL paragraph. The word I-O-CONTROL must begin in Area A, and it must be followed by a period followed by a space.

Each clause within the I-O-CONTROL entry can optionally be separated from the next by a comma or semicolon followed by a space. The clauses, when present, must be specified in the order shown. Clauses can be specified on the same line as the I-O-CONTROL paragraph header, or on separate lines. The I-O-CONTROL entry ends with a period followed by a space.

RERUN Clause

The RERUN clause is syntax-checked, but is treated as documentation.

Assignment-Name: This name can be any user-defined word.

SAME Clause

The SAME clause specifies that two or more files are to use the same main storage area during processing. The files named in a SAME clause need not have the same organization or access.

The following discussion describes only the SAME RECORD AREA and SAME AREA clauses. The SAME SORT AREA and SAME SORT-MERGE AREA clauses are discussed under “SORT/MERGE” on page 491.

The SAME RECORD AREA clause and SAME AREA clause are intended to make efficient use of main storage. However, the virtual storage architecture of the AS/400 system eliminates the need for these clauses, and the clauses are supported for compatibility rather than for performance. Use of the SAME RECORD AREA actually degrades performance.

The SAME RECORD AREA clause specifies that two or more files are to use the same main storage area for processing the current record. All the files can be open at the same time. A record in the shared storage area is considered to be both a record of each opened output file in this SAME RECORD AREA clause, and a logical record of the most recently read input file in this SAME RECORD AREA clause.

More than one SAME RECORD AREA clause can be included in a program; however, the following restriction applies:

- A specific file-name must not appear in more than one SAME RECORD AREA clause.

The SAME AREA clause is syntax-checked, but is treated as documentation. However, the following restrictions apply:

- A specific file-name must not appear in more than one SAME AREA clause.
- If one or more file-names of a SAME AREA clause appear in a SAME RECORD AREA clause, all of the file-names in that SAME AREA clause must appear in that SAME RECORD AREA clause. However, that SAME RECORD AREA clause can contain additional file-names that do not appear in that SAME AREA clause.
- For compatibility, only one of the files for which the SAME AREA clause is specified should be open at one time. This rule takes precedence over the SAME RECORD AREA rule that all the files can be open at the same time.

Note: The SAME RECORD AREA clause allows transfer of data from one file to another with no explicit data manipulation because the input/output record areas of named files are identical, and all are available to the user.

MULTIPLE FILE TAPE Clause

The MULTIPLE FILE TAPE clause is syntax-checked, but is treated as documentation. This clause specifies that two or more files share the same reel of tape. The function is provided by the system through the use of command language. See CRTTAPF, CHGTAPF, and OVRTAPF commands in the *CL Reference*.

COMMITMENT CONTROL Clause

The COMMITMENT CONTROL clause specifies the files that will be placed under commitment control when they are opened. These files will then be affected by the COMMIT and ROLLBACK statements. The COMMIT statement allows the synchronization of changes to data base records while preventing other jobs from modifying those records until the COMMIT is complete. The ROLLBACK statement provides a method of cancelling changes made to data base files when those changes should not be made permanent.

The COMMITMENT CONTROL clause can specify only files assigned to a device type of DATABASE. Files under commitment control may have an organization of sequential, relative or indexed, and may have any access mode valid for a particular organization.

The system locks records contained in files under commitment control when these records are accessed. Records remain locked until released by a `COMMIT` or `ROLLBACK` statement. For more information about record locking for files under commitment control, see “Commitment Control Considerations” on page 247.

Note: Always try to use files in a consistent manner to avoid record locking problems, and to avoid reading records that have not yet been permanently committed to the data base. Typically, a file should either always be accessed under commitment control or never be accessed under commitment control.

Chapter 9. Data Division

Data Division Concepts

The Data Division of a COBOL source program describes all the data to be processed by the object program. Two types of data can be processed: external data and internal data.

External Data

External data is contained in files. A file is a collection of data records existing on some input/output device. A file can be thought of as a group of physical records; it can also be thought of as a group of logical records. The Data Division source statements describe the relationship between physical and logical records. (See the Glossary for definitions of these items.)

A physical record is a unit of data that is treated as an entity when it is moved into or out of auxiliary storage. The size of a physical record is determined by the particular input/output device on which it is stored. The size does not necessarily have a direct relationship to the size or content of the logical information contained in the file.

A logical record is a unit of data whose subdivisions have a logical relationship. A logical record can itself be a physical record (that is, be contained completely in one physical unit of data), or several logical records can be contained within one physical record.

Record description entries, which follow the FD (file description) entry for a specific file, describe the logical records in the file. These entries also describe the category and format of data within each field of the logical record and different values the data might be assigned.

The FD entry specifies the physical aspects of the data such as the size relationship between physical and logical records, the size and name(s) of the logical record(s), and labeling information.

Once the relationship between physical and logical records has been established, only logical records are made available to the COBOL program. Thus, in this manual, a reference to records means logical records unless the term physical records is used.

Internal Data

Program logic can develop additional data within storage. Such data is called internal data.

The concept of logical records applies to internal data as well as to external data. Internal data can thus be grouped into logical records and be defined by a series of record description entries. Items that need not be so grouped can be defined in independent data entries.

Data Relationships

The relationships of all data to be used in a program are defined in the Data Division through a system of level indicators and level-numbers.

A level indicator, together with its descriptive entry, identifies each file description in a program. Level indicators are the highest level of any data hierarchy with which they are associated.

A level-number, together with its descriptive entry, indicates the properties of specific data. Level-numbers can be used to describe a data hierarchy. They can indicate that this data has a special purpose, and while they can be associated with and be subordinate to level indicators, they can also be used independently to describe internal data or data common to two or more programs.

Data Division Organization

The Data Division is divided into three sections: the File Section, the Working-Storage Section, and the Linkage Section. Each section has a specific logical function within a COBOL source program, and each can be omitted from the source program when that logical function is not needed.

Format

```
  DATA DIVISION.
[
| FILE SECTION.
]
[
|  [ file-description-entry,{record-description-entry } . . . ] . . .
                                                                      ]
   [ sort-merge-file-description-entry, { record-description-entry } . . . ] ] . . .
                                                                      ]
[
| WORKING-STORAGE SECTION.
]
[
|  [ data-description-entry ] . . .
]
   [ record-description-entry ] . . . ] . . . ]
[
| LINKAGE SECTION.
]
[
|  [ data-description-entry ] . . .
]
   [ record-description-entry ] . . . ] . . . ]
```

The Data Division must begin with the words `DATA DIVISION` followed by a period and a space.

In the source program, the Data Division sections must appear in the order shown.

Coding Example

```
SEQUENCE        C  A     B
PAGE SERIAL     O
                N
                T
1    3  4    6  7  8     12    16    20    24    28
003010    DATA   DIVISION.
          FILE   SECTION.
000200    FD     FILE-NAME
000300           BLOCK  ------->
000400           RECORD ------->
000500           LABEL  RECORD  ---->
000600           LINAGE ------->
000700           DATA   RECORD  IS  ------->.
000800    01     DESCRIPTION.
000900    WORKING-STORAGE  SECTION.
001000    01     NAME-DESCRIPTION.
001100    01     RECORD-DESCRIPTION.
001200
001300
```

Example Data Division Entries

```
.A 1 B..+....2....+....3....+....4....+....5....+....6....+....7

  DATA DIVISION.
  FILE SECTION.
  FD  INPUT-DATA
      BLOCK CONTAINS 1 RECORDS
      RECORD CONTAINS 80 CHARACTERS
      LABEL RECORDS ARE STANDARD
      DATA RECORDS ARE GEN-INFO SALES-DATA.
  01  GEN-INFO.
      03  EMPLOYEE-NAME.
          05  FIRST-NAME                 PIC X(12).
          05  LAST-NAME                  PIC X(12).
      03  SOC-SEC-NUMBER                 PIC 9(9).
      03  CHECK-SSN
          REDEFINES SOC-SEC-NUMBER        PIC X(9).
      03  AGE                            PIC 99.
      03  BIRTH-DATE.
          05  B-MONTH                    PIC 99.
          05  B-DAY                      PIC 99.
          05  B-YEAR                     PIC 99.
      03  ANNUAL-SALARY                  PIC 9(5)V99.
      03  CHECK-SALARY
          REDEFINES ANNUAL-SALARY         PIC X(7).
 *    THIS REDEFINES WILL BE USED TO SEE IF THE FIELD IS BLANK.
      03  RECORD-ID                      PIC X.
      03  FILLER                         PIC X(31).
  01  SALES-DATA.
      03  SALES-SSN                      PIC 9(9).
      03  SALES-LOCATION                 PIC XX.
          88 MICHIGAN        VALUE IS "MI"
          88 EASTERN-REGION  VALUES ARE "PA" "NY"
          88 HEADQUARTERS    VALUES ARE "BA" THRU "BZ".
      03  TOTAL-COMMISSION               PIC 9(5)V99.
      03  RECORD-CODE                    PIC X.
      03  FILLER                         PIC X(61).
  FD  REPORT-OUT
      LABEL RECORDS ARE OMITTED
      RECORD CONTAINS 132 CHARACTERS
      LINAGE IS 66 LINES
           FOOTING 6 LINES AT TOP 4 LINES AT BOTTOM 4
      DATA RECORD IS PRINT-OUT.
  01  PRINT-OUT                PIC X(132).
  WORKING-STORAGE SECTION.
  01  RECORDS-IN               PIC 9(6)      VALUE ZEROS.
  01  DECLARATIVE-ERRORS       PIC 9(4)      VALUE ZEROS.
  01  EOF-SW                   PIC X         VALUE ZERO.
  01  BAD-DATA-COUNTER         PIC 9(3)      VALUE ZEROS.
  01  CHECK-IT                 PIC XX.
  01  PRINT-FIELDS-EDITED.
      03  FILLER               PIC X(14)     VALUE SPACES.
      03  TOTAL-SALARY         PIC $$$,$$$.99BB.
      03  COMMISSION-COSTS     PIC $**,***,***.99B.
      03  FILLER               PIC X(65)     VALUE ALL "-".
      03  FILLER               PIC X(12)
                               VALUE "---END---JOB".
  01  SALARY-COUNTER           PIC 9(6)V99 VALUE ZEROS.
  01  COMMISSION-COUNTER       PIC 9(6)V99 VALUE ZEROS.
```

File Section

The File Section contains a description of all externally stored data (FD) and a description of each sort-merge file (SD) used in the program.

The File Section must begin with the header FILE SECTION followed by a period. The File Section contains file description entries and sort-merge file description entries. Each entry is followed by its associated record description entry (or entries).

In a COBOL program, the file description entries (beginning with the level indicators FD and SD) represent the highest level of organization in the File Section. The file description entry provides information about the physical structure and identification

of a file, and gives the record-name(s) associated with that file. For further description of the format and the clauses required in a file description entry, see “File Description Entry” on page 300. See “Data Division–SORT/MERGE” on page 493 for a complete discussion of the sort-merge file description entry.

The record description entry consists of a set of data description entries that describe the records contained within a particular file. More than one record description entry can be specified; each is an alternative description of the same storage area. For the format and the clauses required within the record description entry, see “Data Description” on page 309 in this chapter.

IBM Extension

The record description entry for a file can be specified using the Format 2 `COPY` statement. This allows the field descriptions for a record format to be exactly as defined in `DDS`. Also, programs are easier to write because the record format description is maintained in only one place. See “Format 2 COPY Statement, DDS or DD Formats” on page 219 for further information.

End of IBM Extension

Data areas described in the File Section should not be considered available for processing unless the file containing the data area is open.

Working-Storage Section

The Working-Storage Section can contain description records that are not part of data files but are developed and processed internally. These records are used for report description, counters, and other functions necessary in processing data.

The Working-Storage Section must begin with the section header `WORKING-STORAGE SECTION` followed by a period. The Working-Storage Section contains record description entries and data description entries for noncontiguous data items.

Data elements in the Working-Storage Section that bear a definite hierarchical relationship to one another must be grouped into records structured by level-number.

Noncontiguous items in this section that bear no hierarchical relationship to one another need not be grouped into records provided they do not need to be further subdivided. Instead, they are classified and defined as noncontiguous elementary items. Each is defined in a separate data description entry that begins with the special level-number 77 or level-number 01. The format of the data description entry is the same as the format for the record description entry.

Linkage Section

The Linkage Section describes data made available from another program.

Record description entries and data description entries in the Linkage Section provide names and descriptions, but storage within the program is not reserved because the data area exists elsewhere. Any data description clause can be used to describe items in the Linkage Section with one exception: the `VALUE` clause

cannot be specified for any items other than level-88 items. See “Inter-Program Communication Function” on page 507 for additional information.

File Description Entry

In a COBOL program, the file description entry (FD entry) or the sort-merge file description entry (SD entry) is the highest level of organization in the File Section. Up to 99 FD and SD entries can be defined in a COBOL program.

Format 1–Files[1]

```
[ FD file-name

    [ BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS    } ]
                                                { CHARACTERS } ]

    [ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]

  *********************************************************
  *    LABEL { RECORD IS   } { STANDARD }                 *
  *          { RECORDS ARE } { OMITTED  }                 *
  *     [                                                 *
  *     | VALUE OF user-name-1 IS { data-name-1 }         *
  *     |                        { literal-1   }         *
  *     [                                                 *
  *           [                                     ]     *
  *           | , user-name-2 IS { data-name-2 } ] ... |   *
  *           |                 { literal-2   } ]     |   *
  *           [                                     ]     *
  *********************************************************

    [ DATA { RECORD IS   } data-name-3 [ , data-name-4 ] ... ] .
           { RECORDS ARE }                                    ]

{ record-description-entry } ... ] ...
```

[1] Format 1 of the file description entry is used with (FORMATFILE, DATABASE, DISK, READER, PUNCH, PUNCHPRINT, and PRINT) files.

Format 2–Files (DISKETTE)

```
[ FD file-name

    [ BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS    } ]
                                                { CHARACTERS }

    [ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]

  *********************************************************
  *    LABEL { RECORD IS   } { STANDARD }                 *
  *          { RECORDS ARE } { OMITTED  }                 *
  *     [ VALUE OF user-name-1 IS { data-name-1 }         ] *
  *               [                { literal-1   }       ] *
  *               [ , user-name-2 IS { data-name-2 } ] ... ] *
  *               [                  { literal-2   } ]     ] *
  *********************************************************

    [ DATA { RECORD IS   } data-name-3 [ , data-name-4 ] ... ]
           { RECORDS ARE }

    [ CODE-SET is alphabet-name ] .

{ record-description-entry } ... ] ...
```

Format 3–Files (TAPEFILE)

```
[ FD file-name

    [ BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS    } ]
                                                { CHARACTERS }

    [ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]

      LABEL { RECORD IS   } { STANDARD }
            { RECORDS ARE } { OMITTED  }
  *********************************************************
  *     [                                                 ] *
  *     [ VALUE OF user-name-1 IS { data-name-1 }         ] *
  *               [                { literal-1   }       ] *
  *               [ , user-name-2 IS { data-name-2 } ] ... ] *
  *               [                  { literal-2   } ]     ] *
  *********************************************************

    [ DATA { RECORD IS   } data-name-3 [ , data-name-4 ] ... ]
           { RECORDS ARE }

    [ CODE-SET is alphabet-name ] .

{ record-description-entry } ... ] ...
```

Format 4–Files (PRINTER)

```
[ FD file-name

    [ BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS    } ]
                                                { CHARACTERS } ]

    [ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]

  *********************************************************
  *    LABEL { RECORD IS   } { STANDARD }                 *
  *          { RECORDS ARE } { OMITTED  }                 *
  *                                                       *
  *     [ VALUE OF user-name-1 IS { data-name-1 }         *
  *     [                         { literal-1   }         *
  *                                                       *
  *           [ , user-name-2 IS { data-name-2 } ] ... ]  *
  *           [                  { literal-2   } ]        *
  *                                                       *
  *********************************************************

    [ DATA { RECORD IS   } data-name-3 [ , data-name-4 ] ... ]
           { RECORDS ARE }

    [ LINAGE IS { data-name-5 } LINES [ , WITH FOOTING AT { data-name-6 } ]
                { integer-5   }                           { integer-6   }

    [ , LINES AT TOP { data-name-7 } ] [ , LINES AT BOTTOM { data-name-8 } ] ] .
                     { integer-7   } ]                     { integer-8   } ]

{ record-description-entry } ... ] ...
```

Format 5–Sort or Merge File Description

```
[ SD file-name

   [ RECORD CONTAINS [ integer-1 TO ] integer-2 CHARACTERS ]

   [ DATA { RECORD IS   } data-name-1 [ , data-name-2 ] . . . ] .
          { RECORDS ARE }

{ record-description-entry } . . . ]
```

See “Data Division–SORT/MERGE” on page 493 for a discussion of Format 5.

Format 6–TRANSACTION File

```
SELECT file-name
                                ******************************
     ASSIGN TO assignment-name-1 * [ , assignment-name-2 ] ... *
                                ******************************

     ORGANIZATION IS TRANSACTION

    [ ACCESS MODE IS { SEQUENTIAL                            } ]
                     { DYNAMIC, RELATIVE KEY IS data-name-3 }

    [ FILE STATUS IS data-name-1 { , data-name-5 } ]

    [ CONTROL-AREA is data-name-6 ] .
```

See “File-Control Entry” on page 122 for a discussion of Format 6.

The file description entry must begin with the level indicator FD followed by a space.

The clauses that follow file-name are optional in many cases; the order of their appearance is not significant.

However, at least one record description entry must follow the FD entry. When more than one record description entry is specified, each entry implies a redefinition of the same storage area. The last clause in the FD entry must be immediately followed by a period and a space.

File-Name

The file-name must follow the level indicator, and must be the same as that specified in the associated file control entry.

The file-name must follow the rules of formation for a user-defined word; at least one character must be alphabetic. The file-name must be unique within this program.

BLOCK CONTAINS Clause

This clause is syntax-checked, but is treated as documentation except for tape files.

The BLOCK CONTAINS clause specifies the size of a physical record. When the BLOCK CONTAINS clause is omitted, the compiler assumes that records are not blocked. Thus, this clause can be omitted when each physical record contains only one complete logical record.

Format

```
[ BLOCK CONTAINS [ integer-1 TO ] integer-2 { RECORDS    } ]
                                            { CHARACTERS }
```

Integer-1 and integer-2 must be nonzero unsigned integers.

When neither the CHARACTERS nor RECORDS phrase is specified, the CHARACTERS phrase is assumed.

RECORDS Phrase: When the RECORDS phrase is specified, the physical record size is the number of logical records contained in each physical record.

Note: Maximum record size is 32 767; maximum block size is 32 767. These maximums include any control bytes required for variable blocked records; thus, the maximum size data record for a variable-blocked record is 32 759.

CHARACTERS Phrase: When the CHARACTERS phrase is specified or implied, the physical record size is specified as the number of character positions required to store the physical record no matter what USAGE clause the characters within the data record have.

If only integer-2 is specified, it specifies the exact character size of the physical record. When integer-1 and integer-2 are both specified, they represent, respectively, the minimum and maximum character size of the physical record.

Note: Each variable record contains a 4-byte header and each block contains a 4-byte header when the data is transferred to tape. However, these 4-byte headers are provided by the system and are of no concern to the COBOL user except that the maximum size of a variable record is restricted to 32 759.

When variable records are used, the BLOCK CONTAINS clause specifies the maximum physical record length, while the logical record length for each record is inferred by the compiler from the record name used in a WRITE statement. If an explicit length is required after a READ statement, the user can obtain it through the I-O-FEEDBACK mnemonic-name.

RECORD CONTAINS Clause

The RECORD CONTAINS clause specifies the size of a file's data records.

Format

```
[ RECORD CONTAINS [ integer-3 TO ] integer-4 CHARACTERS ]
```

The RECORD CONTAINS clause is never required because the size of each record is completely defined in the record description entries. When this clause is specified, the following rules apply:

- Integer-3 and integer-4 must be unsigned, nonzero integers.
- When both integer-3 and integer-4 are specified, integer-3 specifies the size of the smallest data record, and integer-4 specifies the size of the largest data record.
- Integer-4 must not be specified alone unless all the records are the same size. If all records are the same size, integer-4 specifies the exact number of characters in the record.
- The record size must be specified as the number of character positions needed to store the record internally; that is, size is specified in terms of the number of bytes occupied internally by the record's characters, regardless of the number of characters used to represent the item within the record. The size of a record is determined according to the rules for obtaining the size of a group item. For

a further description of record size, see “USAGE Clause” on page 322 in this chapter.

Note: When the RECORD CONTAINS clause is omitted, the record lengths are determined by the compiler from the record descriptions. When one of the entries within a record description contains an OCCURS DEPENDING ON clause, the compiler uses the maximum value of the variable length item to calculate the record length.

Programming Note: The system supports variable length physical records only for files on tape. For all other files, the logical records are truncated or padded to the length of the record as defined in the CRTxxxF CL command. User length in the following table is defined as the largest record associated with the given file, as specified by its record description.

Input/Output Type	User Length Less Than File Record Length	User Length Greater Than File Record Length
Input	Truncation	Pad with blanks.
Output	Pad with blanks	Truncation if old file (non-empty); for new (empty files) the larger record length is used.

LABEL RECORDS Clause

The LABEL RECORDS clause specifies whether labels are present or omitted. The LABEL RECORDS clause is required in every FD entry. Format 3 (TAPEFILE) is the only format in which this clause is not treated as documentation.

Format

```
LABEL { RECORD IS   } { STANDARD }
      { RECORDS ARE } { OMITTED  }
```

IBM Extension

The LABEL RECORDS clause can be changed at run time by specifying the REELS parameter of the Override with Tape File (OVRTAPF) CL command. See the *CL Reference* for more information on this command.

End of IBM Extension

STANDARD Phrase: The STANDARD phrase specifies that labels conforming to system specifications exist for this file. This phrase must be specified for files assigned to DISKETTE, DISK, and DATABASE. (See “FILE-CONTROL Paragraph” on page 281.)

OMITTED Phrase: The OMITTED phrase specifies that no labels exist for this file. This phrase must be specified for files assigned to READER, PUNCHPRINT, PRINT, and PRINTER. (See “FILE-CONTROL Paragraph” on page 281.)

Notes:

1. Card devices are not supported by System/38-Compatible COBOL, even though the devices are accepted by the syntax checker.

VALUE OF Clause

The VALUE OF clause is syntax-checked, but is treated as documentation. It specifies the description of an item in the label records associated with this file.

Format

```
[ VALUE OF user-name-1 IS { data-name-1 }
                          { literal-1   }

  [ , user-name-2 IS { data-name-2 } ] ... ]
                     { literal-2   }
```

User-name: This name follows the rules for the formation of a user-defined word.

DATA RECORDS Clause

The DATA RECORDS clause specifies the names of data records associated with this file. The DATA RECORDS clause is never required.

Format

```
[ DATA { RECORD IS   } data-name-3 [ , data-name-4 ] . . . ]
       { RECORDS ARE }
```

Data-name-3 and data-name-4 are the names of data records and must have 01 level-number record descriptions that have the same name associated with them.

The specification of more than one data-name indicates that this file contains more than one type of data record. Two or more record descriptions for this file occupy the same storage area. These records need not have the same description or length. The order in which the data-names are listed is not significant.

LINAGE Clause

The LINAGE clause specifies the depth of a logical page in terms of the number of lines. This clause also optionally specifies the line number at which the footing area begins, as well as the top and bottom margins of the logical page.

At run time, the printer file being used determines the physical page size. This information is used to issue appropriate space and eject commands to produce the logical page as defined in the LINAGE clause. Thus, the logical page can contain multiple physical pages, or one physical page can contain multiple logical pages.

Format

```
[ LINAGE IS { data-name-5 } LINES [ , WITH FOOTING AT { data-name-6 } ] ]
            { integer-5   }       [                   { integer-6   } ]

[ , LINES AT TOP { data-name-7 } ] [ , LINES AT BOTTOM { data-name-8 } ] ]
                 { integer-7   } ]                     { integer-8   } ] ]
```

The LINAGE clause can be specified only for files assigned to the device PRINTER. See "FILE-CONTROL Paragraph" on page 281.

All integers must be unsigned. All data-names must be described as unsigned integer data items.

Integer-5 or the value in data-name-5 specifies the number of lines that can be written and/or spaced on this logical page. The area of the page that these lines represent is called the page body. The value must be greater than zero.

WITH FOOTING Phrase: Integer-6 or the value in data-name-6 specifies the first line number of the footing area within the page body. The footing line number must be greater than zero, but it must not be greater than the number for the last line of the page body. The footing area extends between those two lines. If this phrase is not specified, the assumed value is equal to that of the page body (integer-5 or data-name-5).

LINES AT TOP Phrase: Integer-7 or the value in data-name-7 specifies the number of lines in the top margin of the logical page. The value of integer-7 or data-name-7 can be zero. If this phrase is not specified, zero is assumed.

LINES AT BOTTOM Phrase: Integer-8 or the value in data-name-8 specifies the number of lines in the bottom margin of the logical page. The value of integer-8 or data-name-8 can be zero. If this phrase is not specified, zero is assumed.

Figure 69 on page 308 illustrates the use of each phrase of the LINAGE clause.

LINAGE Clause Considerations: The logical page size specified in the LINAGE clause is the sum of all values specified in each phrase except the FOOTING phrase. If the LINES AT TOP and/or the LINES AT BOTTOM phrases are zero, each logical page immediately follows the preceding logical page with no additional spacing provided.

At the time an OPEN OUTPUT statement is processed, the values of integer-5, integer-6, integer-7, and integer-8 are used to determine the page body, first footing line, top margin, and bottom margin of the logical page for this file. These values are then used for all logical pages printed for this file when the program is run.

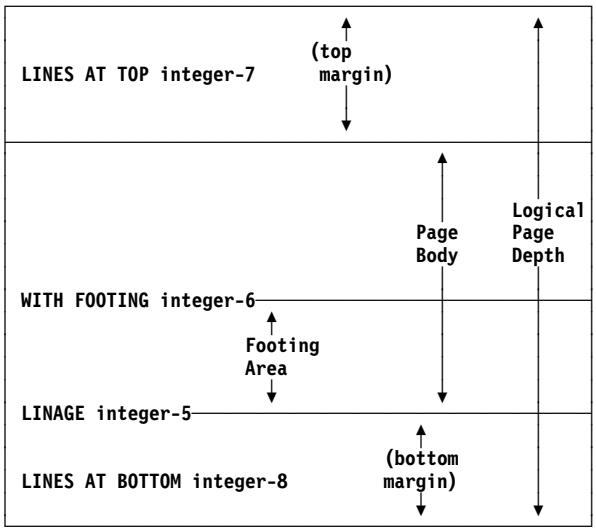


Figure 69. LINAGE Clause and Logical Page Depth

If the FOOTING phrase is specified and the value of data-name-6 or integer-6 is equal to the LINAGE value of data-name-5 or integer-5, one line (the last line of the logical page) is available for footing information. If the FOOTING phrase is not specified, no footing area is provided at the end of the logical page, even though the default FOOTING value is data-name-5 or integer-5.

Data-name-5, data-name-6, data-name-7, and data-name-8 cause the following actions to take place:

- Their values at the time an OPEN OUTPUT is processed are used to determine the page body, the first footing line, the top margin, and the bottom margin for the first logical page only.
- Their values at the time a WRITE ADVANCING statement causes page ejection are used to determine the page body, first footing line, top margin, and bottom margin for the next succeeding logical page only.

LINAGE-COUNTER Special Register: For each FD entry containing a LINAGE clause, a separate LINAGE-COUNTER special register is generated. LINAGE-COUNTER is initialized to one when an OPEN statement for this file is processed. LINAGE-COUNTER is automatically modified by any WRITE statement for this file.

If more than one FD has a LINAGE clause, then when LINAGE-COUNTER special register is referred to in the PROCEDURE DIVISION, the user must qualify each LINAGE-COUNTER with its related file-name. For example, LINAGE-COUNTER OF FILE-A.

The value in LINAGE-COUNTER at any given time is the line number at which the device is positioned within the current page. LINAGE-COUNTER can be referred to in Procedure Division statements; LINAGE-COUNTER must not be modified by these statements.

CODE-SET Clause

The CODE-SET clause is valid only for files assigned to TAPEFILE or DISKETTE. This clause specifies the character code that is used to represent data on a magnetic tape file or diskette file.

Format

```
[ CODE-SET IS alphabet-name ]
```

When the CODE-SET clause is specified, the following rules apply:

- Alphabet-name identifies the character code convention that is used to represent data on the input/output device.
- All data in this file must have USAGE DISPLAY.
- If signed numeric data is present, it must be described by the SIGN IS SEPARATE clause.
- Alphabet-name must be defined in the SPECIAL-NAMES paragraph as STANDARD-1 for ASCII encoded files or as NATIVE for EBCDIC encoded files.

The CODE-SET clause specifies the algorithm for converting the character codes on the input/output medium from or to the internal EBCDIC character set.

IBM Extension

If the CODE-SET clause is omitted, the CODE parameter of the Create Diskette File (CRTDKTF) or the Create Tape File (CRTTAPF) CL command is used.

The CODE-SET clause can be changed at run time by specifying the CODE parameter on the Override with Diskette File (OVRDKTF) or the Override with Tape File (OVRTAPF) CL command. See the *CL Reference* for more information on these commands.

End of IBM Extension

Data Description

All the data used in a COBOL program is described using a uniform system of representation. The basic concepts of data description are discussed in this chapter, as well as the actual COBOL clauses used to describe data.

Data Description Concepts

Most of the data processed by a COBOL program is presented in hierarchically arranged records. This is necessary because most data must be divided into subdivisions for processing. To subdivide such records, COBOL uses a hierarchical concept of levels.

For example, in a department store's customer file, one complete record could contain all data pertaining to one customer. Subdivisions within that record could be: customer name, customer address, account number, department number of

sale, unit amount of sale, dollar amount of sale, previous balance, and other pertinent information.

Level Concepts

Because records must be divided into logical subdivisions, the concept of levels is inherent in the structure of a record. Once a record has been subdivided, it can be further subdivided to provide more detailed data references.

The basic subdivisions of a record (that is, those fields that are not further subdivided) are called elementary items. Thus, a record can be made up of a series of elementary items, or it can itself be an elementary item.

It might be necessary to refer to a set of elementary items. Thus, elementary items can be combined into group items. Groups can be combined into a more inclusive group that contains two or more subgroups. Thus, within one hierarchy of data items, an elementary item can belong to more than one group item.

Level-Numbers

A system of level-numbers specifies the organization of elementary and group items into records. Special level-numbers are also used to identify data items used for special purposes.

Each group and elementary item in a record requires a separate entry, and each must be assigned a level-number. The following level-numbers are used to structure records:

01	This level-number specifies the record itself and is the most inclusive level-number possible. A level-01 entry can be either a group item or an elementary item.
02-49	These level-numbers specify group and elementary items within a record. Less inclusive data items are assigned higher (not necessarily consecutive) level-numbers.

A group item includes all group and elementary items following it until a level-number less than or equal to the level-number of this group is encountered.

All elementary or group items immediately subordinate to one group item must be assigned identical level-numbers that are higher than the level-number of this group item.

IBM Extension

Elementary items or group items that are immediately subordinate to one group item can have unequal level-numbers. For example, group item A consists of items B, C, and D:

```
01  A.
    05  B PIC X(4).
    04  C PIC X(20).
    02  D PIC 99.
```

IBM does not recommend such coding practices, and this extension is provided only for compatibility.

End of IBM Extension

Figure 70 illustrates the level-number concept. Notice that all groups immediately subordinate to the level-01 entry have the same level-number. Notice also that elementary items from different subgroups do not necessarily have the same level-number, and that elementary items can be specified at any level within the hierarchy. Figure 70 shows the COBOL record-description entry in the left portion of the figure; it shows the subdivision of the entry in the right portion of the figure.

Note: Level-numbers 01 through 09 can also be written as 1 through 9.

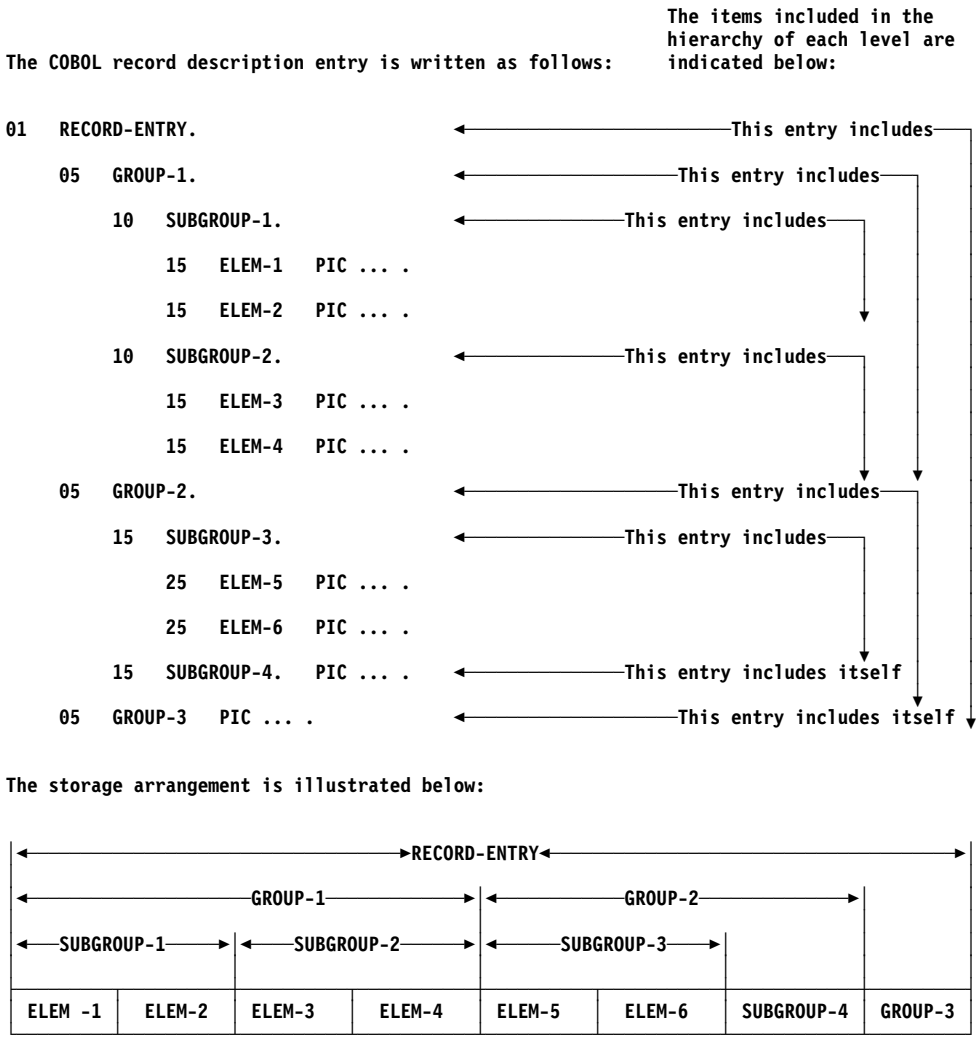


Figure 70. Level-Number Concepts

Special Level-Numbers

Special level-numbers identify items that do not structure a record. The following are special level-numbers:

66 This level-number identifies elementary or group items described by a RENAMES clause. Such items regroup previously defined data items.

77 This level-number identifies independent data description entries in the Working-Storage or Linkage Section. These items are not subdivisions of other items, and are not themselves subdivided.

88 This level-number identifies any condition-name entry that is associated with a particular value or values of a conditional variable. An example is given under "VALUE Clause" in this chapter.

Note: Level-77 and level-01 entries in the Working-Storage Section and Linkage Section must be given unique data-names because neither can be qualified. If subordinate data-names can be qualified, they need not be unique.

Indentation

Successive data description entries can begin in the same column as preceding entries, or they can be indented according to level-number. Indentation is useful for documentation, but it does not affect the action of the compiler.

Classes of Data

All data used in a COBOL program can be divided into four classes and six categories. Every elementary item in a program belongs to one of the classes as well as one of the categories. Every group item belongs to the alphanumeric class even if the subordinate elementary items belong to another class and category. Figure 71 on page 313 shows the relationship of data classes and categories.

IBM Extension

Boolean data is an IBM extension that provides a means of modifying and passing the values of the indicators associated with the display screen formats. A Boolean value of 0 is the off status of the indicator, and a Boolean value of 1 is the on status of the indicator.

A Boolean literal contains a single 0 or 1 and is enclosed in quotes and immediately preceded by an identifying B. The Boolean literal is defined as either B"0" or B"1". A Boolean character occupies one byte. The figurative constant ZERO can be used as a Boolean literal, and the reserved word ALL is valid with a Boolean literal.

End of IBM Extension

Level of Item	Class	Category
Elementary	Alphabetic	Alphabetic
	Boolean	Boolean
	Numeric	Numeric
	Alphanumeric	Numeric edited Alphanumeric edited Alphanumeric
Group	Alphanumeric	Alphabetic Boolean Numeric Numeric edited Alphanumeric edited Alphanumeric

Figure 71. Classes and Categories of Data

Standard Alignment Rules

The standard alignment rules for positioning data in an elementary item depend on the data category of the receiving item (that is, the item into which the data is placed).

Numeric Items: When a numeric item is the receiving item, the following rules apply:

- The data is aligned on the assumed decimal point (`PICTURE` character `V`) and, if necessary, truncated or padded with zeros. (An assumed decimal point is one that has logical meaning but does not exist as a character in the data.)
- If an assumed decimal point is not explicitly specified, the receiving item is treated as though an assumed decimal point is specified immediately to the right of the field. The data is then treated as in the preceding rule.

Numeric Edited Items: The data is aligned on the decimal point and, if necessary, truncated or padded with zeros at either end, except when editing causes replacement of leading zeros.

Alphanumeric, Alphanumeric Edited, Alphabetic: For these data categories, the following rules apply:

- The data is aligned at the leftmost character position and, if necessary, truncated or padded with spaces at the right.
- If the `JUSTIFIED` clause is specified for this receiving item, the above rule is modified as described in the `JUSTIFIED` clause.

Standard Data Format

COBOL makes data description as machine independent as possible. For this reason, the properties of the data are described in a standard data format rather than a machine-oriented format.

The standard data format uses the decimal system to represent numbers no matter what base is used by the system. The nonnumeric data can contain any characters that are in the native character set, that is, nonnumeric data is not limited to just the COBOL character set or the nonnumeric COBOL characters.

Character-String and Item Size

In COBOL, the size of an elementary item is determined through the number of character positions specified in its PICTURE character-string. In storage, however, the size is determined by the actual number of bytes the item occupies as determined by the combination of its PICTURE character-string and its USAGE clause.

When an arithmetic item is moved from a longer field to a shorter one, the compiler truncates the data to the number of characters represented in the shorter item's PICTURE character-string.

For example, if a sending field with PICTURE S99999 and containing the value +12345 is moved to a COMPUTATIONAL receiving field with PICTURE S99, the data is truncated to +45.

Signed Data

There are two categories of algebraic signs used in COBOL: operational and editing.

Operational Signs

Operational signs (+, -) are associated with signed numeric items and indicate their algebraic properties. The internal representation of an algebraic sign depends on the item's USAGE clause and optionally upon its SIGN clause. Zero is considered a unique value regardless of the operational sign. An unsigned field is always assumed to be positive or zero.

Editing Signs

Editing signs are associated with numeric edited items. Editing signs are PICTURE symbols (+, -, CR, DB) that identify the sign of the item in edited output.

Record Description Entry

A record description entry consists of one or more data description entries. The maximum length of a record description entry is restricted to 32 767 bytes.

Data Description Entry

A data description entry specifies the characteristics of a particular data item. The maximum length for any item that is not otherwise restricted is 32 767 bytes. The general formats are:

Format 1

```
level-number { data-name-1 }
             { FILLER      }

      [ REDEFINES data-name-2 ]

      [ { PICTURE } IS character-string ]
      [ { PIC     }                     ]

      [                 {   DISPLAY           } ]
      [                 {   COMPUTATIONAL     } ]
      [                 {   COMP              } ]
      [                 {                     } ]
      [ USAGE IS ]      {   COMPUTATIONAL-3   } ]
      [                 {   COMP-3            } ]
      [                 {   COMPUTATIONAL-4   } ]
      [                 {   COMP-4            } ]
      [                 {                     } ]
      [                 {   INDEX             } ]

      [ [ SIGN IS ] { LEADING  } [ SEPARATE CHARACTER ] ]
      [             { TRAILING }                        ]

      [ OCCURS { integer-1 TO integer-2 TIMES DEPENDING ON data-name-3 } ]
      [        { integer-2 TIMES                                       } ]

               [ { ASCENDING  } KEY IS data-name-4 [ , data-name-5 ] . . . ] . . .
               [ { DESCENDING }                                            ]

               [ [ INDEXED BY index-name-1 [ , index-name-2 ] . . . ] ]

     *********************************
     * [ { SYNCHRONIZED } [ LEFT  ] ] *
     * [ { SYNC         } [ RIGHT ] ] *
     *********************************

      [ { JUSTIFIED } RIGHT ]
      [ { JUST      }       ]

      [ BLANK WHEN ZERO ]

      [ VALUE IS literal ] .
```

Format 2

```
66 data-name-1 RENAMES data-name-2 [ { THROUGH } data-name-3 ] .
                                   [ { THRU    }             ]
```

Format 3

```
88 condition-name { VALUE IS   } literal-1 [ { THROUGH } literal-2 ]
                  { VALUES ARE }           [ { THRU    }         ]

     [ literal-3 [ { THROUGH } literal-4 ] ] . . . .
                 [ { THRU    }           ]
```

Format 4–Boolean Data

```
level-number { data-name-1 }
             { FILLER      }

      [ REDEFINES data-name-2 ]

      [ { PICTURE } IS 1 ]
      [ { PIC     }      ]

      [ [ USAGE IS ] DISPLAY ]

      [ OCCURS { integer-1 TO integer-2 TIMES DEPENDING ON data-name-3 } ]
               { integer-2 TIMES                                        }


               [ INDEXED BY index-name-1 [ , index-name-2 ] . . . ] ]

      [ { INDICATOR  }                ]
      [ { INDICATORS } integer-3      ]
      [ { INDIC      }                ]

     **********************************
     * [                            ] *
     * [ { SYNCHRONIZED } [ LEFT  ] ] *
     * [ { SYNC         } [ RIGHT ] ] *
     * [                            ] *
     *                                *
     * [ { JUSTIFIED } RIGHT ]        *
     * [ { JUST      }       ]        *
     **********************************

      [ VALUE IS Boolean-literal ] .
```

Format 1

This format is used for record description entries in all sections and for level-77 entries in the Working-Storage and Linkage Sections. The following rules apply:

- Level-number can be any number from 01 through 49 or 77. Level-numbers from 01 through 09 can be coded as 1 through 9.

- The clauses can be written in any order, with two exceptions:
  - The data-name/FILLER clause must immediately follow the level-number.
  - When specified, the REDEFINES clause must immediately follow the data-name clause.
- The PICTURE clause must be specified for every elementary item except index data items.
- The BLANK WHEN ZERO, JUSTIFIED, PICTURE, and SYNCHRONIZED clauses are valid only for elementary items.
- Either a space, or a comma or a semicolon followed by a space, must separate clauses.
- Each entry must end with a period followed by a space.

Format 2

This format regroups previously defined items. The following rules apply:

- A level-66 entry cannot rename another level-66 entry, nor can it rename a level-01, level-77, or level-88 entry.
- All level-66 entries associated with one record must immediately follow the last data description entry in that record.
- The entry must end with a period followed by a space.

See “RENAMES Clause” later in this chapter for a further description.

Format 3

This format describes condition-names. A condition-name is a user-specified name that associates value(s) and/or a range(s) of values with a conditional variable.

A conditional variable is a data item that can assume one or more values that can, in turn, be associated with a condition-name. The following rules for condition-name entries apply:

- Any entry beginning with level-number 88 is a condition-name entry.
- The condition-name entries associated with a particular conditional-variable must immediately follow the conditional variable entry. The conditional variable can be any elementary data description entry except another condition-name, index data item, or level-66 entry.
- A condition-name can be associated with a group item data description entry. The following rules apply:
  - The condition-name value must be specified as a nonnumeric literal or figurative constant.
  - The size of the condition-name value must not exceed the sum of the sizes of all the elementary items within the group.
  - No element within the group may contain a JUSTIFIED or SYNCHRONIZED clause.
  - No USAGE other than USAGE IS DISPLAY may be specified within the group.
- Condition-names can be specified both at the group level and at subordinate levels within the group.

- The relation test implied by the definition of a condition-name at the group level is processed in accordance with the rules for comparison of nonnumeric operands regardless of the nature of elementary items within the group.
- Either a space or a comma or a semicolon followed by a space, must separate successive operands.
- Each entry must end with a period followed by a space.

Examples of both elementary and group condition-name entries are given under "VALUE Clause" on page 328 in this chapter.

Format 4–Boolean Data

See "Data Description Entry–Boolean Data" on page 94 for a discussion of this format.

Level-Numbers

The level-number specifies the hierarchy of data within a record and also identifies special-purpose data entries.

Format

```
level-number
```

The following rules for level-numbers apply:

- A level-number begins a data description entry, a regrouped item, or a condition-name entry.
- Level-numbers 01 and 77 must begin in Area A.
- Level-numbers 02-49, 66, and 88 can begin in either Area A or Area B and must be followed by a space.
- Single-digit level-numbers 1 through 9 can be substituted for level-numbers 01 through 09.

Data-Name or FILLER Clause

A data-name explicitly identifies the data being described; the keyword `FILLER` specifies an item that is never explicitly referenced in the program.

Format

```
data-name
FILLER
```

In a data description entry, either the data-name or the keyword `FILLER` must be the first word following the level-number. The data-name identifies a data item by referring to the field, not to a particular value. This data item can assume a number of different values during the course of a program.

A data-name can begin anywhere in Area B. A data-name must contain at least one alphabetic character.

Entries at level-numbers 01 and 77 in the Working-Storage and Linkage Sections cannot be qualified, and therefore require unique data-names. Subordinate data-names that can be qualified do not require unique data-names.

The keyword FILLER specifies an elementary item in a record that is never explicitly referred to. The word FILLER can be written anywhere in Area B.

In a MOVE CORRESPONDING statement, an ADD CORRESPONDING statement, or a SUBTRACT CORRESPONDING statement, FILLER items are ignored.

IBM Extension

A FILLER item can be used as a group item definition. Subordinate data items can then be referenced by the appropriate data-name.

End of IBM Extension

REDEFINES Clause

The REDEFINES clause allows the same storage area to be described by different data description entries.

Format

```
level-number data-name-1 REDEFINES data-name-2
```

Level-number and data-name-1 are not part of the REDEFINES clause itself, and are included in the format only for clarity.

If specified, the REDEFINES clause must be the first entry following data-name-1.

The level-number of data-name-1 and data-name-2 must be identical and must not be level-66 or level-88.

Data-name-1 is the redefining item and is an alternative description for the data-name-2 area.

Data-name-2 is the redefined item.

Implicit redefinition is assumed when more than one level-01 entry subordinate to an FD entry is written. In such level-01 entries, the REDEFINES clause must not be specified.

Redefinition begins at data-name-2 and ends when a level-number less than or equal to that of data-name-2 is encountered. No entry having a level-number numerically lower than those of data-name-1 and data-name-2 can occur between these entries.

In the following example, A is data-name-2, and B is data-name-1. Redefinition begins with B and includes the two subordinate items B-1 and B-2. Redefinition ends when the level-05 item C is encountered.

```
05 A PICTURE X(6).
05 B REDEFINES A.
     10 B-1  PICTURE X(2).
     10 B-2  PICTURE 9(4).
05 C PICTURE 99V99.
```

The data description entry for data-name-2 cannot contain a REDEFINES clause or an OCCURS clause. However, data-name-2 can itself be subordinate to an item that contains either clause. If data-name-2 is subordinate to an OCCURS clause, it must not be subscripted or indexed in the REDEFINES clause.

The redefined item, the redefining item, and any items subordinate to them cannot contain an OCCURS DEPENDING ON clause.

When data-name-1 is specified with a level-number other than 01, it must specify a storage area of the same size as data-name-2.

Multiple redefinitions of the same storage area are permitted. The entries giving the new descriptions of the storage area must immediately follow the description of the redefined area without intervening entries that define new character positions. Multiple redefinitions must all use the data-name of the original entry that defined this storage area. For example:

```
05 A PICTURE 9999.
05 B REDEFINES A PICTURE 9V999.
05 C REDEFINES A PICTURE 99V99.
```

Data-name-1 and any subordinate entries must not contain any VALUE clauses. This rule does not apply to condition-name entries.

Data items within an area can be redefined without their lengths being changed. For example:

```
05 NAME-2.
   10 SALARY PICTURE XXX.
   10 SO-SEC-NO PICTURE X(9).
   10 MONTH PICTURE XX.
05 NAME-1 REDEFINES NAME-2.
   10 WAGE PICTURE XXX.
   10 EMP-NO PICTURE X(9).
   10 YEAR PICTURE XX.
```

Data items can also be rearranged within an area. For example:

```
05 NAME-2.
   10 SALARY PICTURE XXX.
   10 SO-SEC-NO PICTURE X(9).
   10 MONTH PICTURE XX.
05 NAME-1 REDEFINES NAME-2.
   10 EMP-NO PICTURE X(6).
   10 WAGE PICTURE 999V999.
   10 YEAR PICTURE XX.
```

When an area is redefined, all descriptions of the area are always in effect; that is, redefinition does not cause any data to be erased and does not supersede the previous description. Thus, if B REDEFINES A has been specified, either of the two procedural statements MOVE X TO B and MOVE Y TO A could be processed at any point in the program.

In the first case, the area described as B would assume the value of X. In the second case, the same physical area (described now as A) would assume the value of Y. If the second statement is processed immediately after the first, the value of Y replaces the value of X in the one storage area.

The USAGE clause of a redefining data item need not be the same as that of a redefined item. This does not, however, cause any change in existing data. For example:

```
05 B PICTURE 99 USAGE DISPLAY VALUE 8.
05 C REDEFINES B PICTURE S99 USAGE
     COMPUTATIONAL-4.
05 A PICTURE S99 USAGE COMPUTATIONAL-4.
```

The bit configuration of the DISPLAY value 8 (held in B) is 1111 0000 1111 1000. Redefining B does not change the bit configuration of the data in the storage area. Therefore, the two statements, ADD B TO A and ADD C TO A give different results. In the first case, the value 8 is added to A (because B has USAGE DISPLAY). In the second statement, the value -48 is added to A (because C has USAGE COMPUTATIONAL-4), and the bit configuration (truncated to two decimal digits) in the storage area has the binary value -48.

Unexpected results can occur when a redefining item is moved to a redefined item (that is, if B REDEFINES C and the statement MOVE B TO C is processed). Unexpected results can also occur when a redefined item is moved to a redefining item (from the previous example, unexpected results occur if the statement MOVE C TO B is processed).

The REDEFINES clause can be specified for an item within the scope of any area being redefined (that is, an item subordinate to a redefined item). For example:

```
05 REGULAR-EMPLOYEE.
   10 LOCATION PICTURE A(8).
   10 GRADE PICTURE X(4).
   10 SEMI-MONTHLY-PAY PICTURE
      9999V99.
   10 WEEKLY-PAY REDEFINES
      SEMI-MONTHLY-PAY
      PICTURE 999V999.

05 TEMPORARY-EMPLOYEE REDEFINES
   REGULAR-EMPLOYEE.
   10 LOCATION PICTURE A(8).
   10 FILLER PICTURE X(6).
   10 HOURLY-PAY PICTURE 99V99.
```

The REDEFINES clause can also be specified for an item subordinate to a redefining item. For example:

```
05 REGULAR-EMPLOYEE.
   10 LOCATION PICTURE A(8).
   10 GRADE PICTURE X(4).
   10 SEMI-MONTHLY-PAY
      PICTURE 999V999.

05 TEMPORARY-EMPLOYEE REDEFINES
   REGULAR-EMPLOYEE.
   10 LOCATION PICTURE A(8).
   10 FILLER PICTURE X(6).
   10 HOURLY-PAY PICTURE 99V99.
   10 CODE-H REDEFINES HOURLY-PAY
      PICTURE 9999.
```

USAGE Clause

The USAGE clause specifies the format of a data item in storage. The USAGE clause can be specified for an entry at any level. However, if it is specified at the group level, it applies to each elementary item in the group. The usage of an elementary item cannot contradict the explicit usage of a group to which the elementary item belongs.

The USAGE clause specifies the format in which data is represented in storage. Consideration must be given to how the data is used in the Procedure Division.

Format

```
                 {   DISPLAY           }
                 {   COMPUTATIONAL     }
                 {   COMP              }
                 {                     }
[ USAGE IS ]     {   COMPUTATIONAL-3   }
                 {   COMP-3            }
                 {   COMPUTATIONAL-4   }
                 {   COMP-4            }
                 {                     }
                 {   INDEX             }
```

When the USAGE clause is not specified at either the group or elementary level, USAGE IS DISPLAY is assumed.

DISPLAY Phrase

The DISPLAY phrase can be explicit or implicit. It specifies that the data item is stored in character form, one character per eight-bit byte. This corresponds to the form in which information is represented for keyboard input or for printed output. USAGE IS DISPLAY is valid for the following types of items:

- Alphabetic
- Alphanumeric
- Alphanumeric edited
- Numeric edited
- Boolean
- Zoned decimal (numeric).

Alphabetic, alphanumeric, alphanumeric edited, numeric edited, and Boolean items are discussed under “PICTURE Clause” on page 332 later in this chapter.

Zoned Decimal Items: These items are sometimes referred to as external decimal items. Each digit of a number is presented by a single byte. The four high-order bits of each byte are zone bits; the four high-order bits of the low-order byte represent the sign of the item. If the number is positive, these four bits contain a hexadecimal F. If the number is negative, these four bits contain a hexadecimal D. The four low-order bits of each byte contain the value of the digit. When zoned decimal items are used for computations, the compiler processes the necessary conversions. The maximum length of a zoned decimal item is 18 digits.

The PICTURE character-string of a zoned item can contain only 9s, the operational sign symbol S, the assumed decimal point V, and one or more Ps.

Examples of zoned decimal items are shown in Figure 72 on page 325.

Computational Phrases

The term computational refers to the following phrases of the USAGE clause:

- COMPUTATIONAL or COMP (packed decimal)

IBM Extension

- COMPUTATIONAL-3 or COMP-3 (packed decimal).
- COMPUTATIONAL-4 or COMP-4 (binary).

End of IBM Extension

A computational item represents a value to be used in arithmetic operations and must be numeric. If the USAGE of a group item is described with any of these options, it is the elementary items within the group that have this usage. The group itself is considered nonnumeric and cannot be used in numeric operations. The maximum length of a computational item is 18 decimal digits.

The PICTURE of a computational item can contain only:

9 (one or more numeric character positions)
S (one operational sign)
V (one implied decimal point)
P (one or more decimal scaling positions).

The COMPUTATIONAL phrase is specified for packed decimal items. Such an item appears in storage as two digits per byte, with the sign contained in the four rightmost bits of the rightmost byte. A packed decimal item can contain any of the digits 0 through 9 plus a sign. If the PICTURE of a packed decimal item does not contain an S, the sign position is occupied by a bit configuration that is interpreted as positive.

IBM Extension

The COMPUTATIONAL-3 phrase is specified for packed decimal items and is considered by the compiler to be equivalent to the COMPUTATIONAL phrase.

The COMPUTATIONAL-4 phrase is specified for binary data items. Such items have decimal equivalents consisting of the decimal digits 0 through 9, plus a sign.

The amount of storage occupied by a binary data item depends on the number of decimal digits defined in its PICTURE clause:

Digits in PICTURE Clause	Storage Occupied
1 through 4	2 bytes
5 through 9	4 bytes
10 through 18	8 bytes

The leftmost bit of the storage area is the operational sign.

End of IBM Extension

Examples of packed decimal and binary items are shown in Figure 72 on page 325.

INDEX Phrase

The USAGE IS INDEX clause specifies that the data item named has an indexed format and, therefore, is an index data item. The index data item is an elementary item that can be used to save index-name values for future reference.

The USAGE IS INDEX clause is described in detail under “TABLE HANDLING” on page 469.

ITEM	DESCRIPTION	VALUE	INTERNAL REPRESENTATION*
Zoned Decimal	PIC S9999 DISPLAY	+1234	F1 F2 F3 F4
		-1234	F1 F2 F3 D4
		1234	F1 F2 F3 F4
	PIC 9999 DISPLAY	+1234	F1 F2 F3 F4
		-1234	F1 F2 F3 F4
		1234	F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN LEADING	+1234	F1 F2 F3 F4
		-1234	D1 F2 F3 F4
		1234	F1 F2 F3 F4
	PIC S9999 DISPLAY SIGN TRAILING SEPARATE	+1234	F1 F2 F3 F4 4E
		-1234	F1 F2 F3 F4 60
		1234	F1 F2 F3 F4 4E
	PIC S9999 DISPLAY SIGN LEADING SEPARATE	+1234	4E F1 F2 F3 F4
		-1234	60 F1 F2 F3 F4
		1234	4E F1 F2 F3 F4
Packed Decimal	PIC S9999 {COMP }	+1234	01 23 4F
	{COMP-3}	-1234	01 23 4D
	PIC 9999 {COMP }	+1234	01 23 4F
	{COMP-3}	-1234	01 23 4F
Binary	PIC S9999 COMP-4	+1234	04 D2
		-1234	FB 2E
	PIC 9999 COMP-4	+1234	04 D2
		-1234	04 D2

* The internal representation of each byte is shown as two hex digits.
The bit configuration for each digit is as follows:

Hex Digit	Bit Configuration	Hex Digit	Bit Configuration
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

NOTES:

1. The leftmost bit of a binary number represents the sign: 0 is positive, 1 is negative.
2. Negative binary numbers are represented in twos complement form.
3. Hex 4E represents the EBCDIC character +, Hex 60 represents the EBCDIC character -.
4. Specification of SIGN TRAILING (without the SEPARATE CHARACTER option) is the equivalent of the standard action of the compiler.

Figure 72. Internal Representation of Numeric Items

SIGN Clause

The SIGN clause specifies the position and mode of representation of the operational sign for a numeric entry.

Format

```
[ [ SIGN IS ] { LEADING  } [ SEPARATE CHARACTER ] ]
              { TRAILING }
```

The SIGN clause can be specified only for a signed numeric data description entry (that is, one whose PICTURE character-string contains an S), or for a group item that contains at least one such elementary entry. USAGE IS DISPLAY must be specified either explicitly or implicitly.

Only one SIGN clause can apply to any one data description entry. The SIGN clause is required only when an explicit description of the properties and/or position of the operational sign is necessary.

The SIGN clause defines the position and mode of representation of the operational sign for the numeric data description entry to which it applies, or for each signed numeric data description entry subordinate to the group to which it applies.

If the SEPARATE CHARACTER phrase is not specified, then:

- The operational sign is presumed to be associated with the leading or trailing digit position (whichever is specified) of the elementary numeric data item.
- The character S in the PICTURE character-string is not counted in determining the size of the item (in terms of standard data format characters).

If the SEPARATE CHARACTER phrase is specified, then:

- The operational sign is presumed to be the leading or trailing character position (whichever is specified) of the elementary numeric data item. This character position must be occupied by an operational sign and is not considered to be a digit position.
- The character S in the PICTURE character string is counted in determining the size of the data item (in terms of standard data format characters).
- + is the character used for the positive operational sign.
- – is the character used for the negative operational sign.

Every numeric data description entry whose PICTURE contains the symbol S is a signed numeric data description entry. If the SIGN clause is also specified for such an entry and conversion is necessary for computations or comparisons, the conversion takes place automatically.

If no SIGN clause is specified for a signed numeric data description entry, the position and mode of representation for the operational sign is determined as explained in the USAGE clause description.

OCCURS Clause

The OCCURS clause specifies tables whose elements can be referred to by indexing or subscripting. This clause is described under “Data Division–Table Handling” on page 476.

INDICATOR Clause

The INDICATOR clause is discussed under “Data Description Entry–Boolean Data” on page 94.

SYNCHRONIZED Clause

The SYNCHRONIZED clause specifies the alignment of an elementary item on a proper boundary in storage.

The SYNCHRONIZED clause is syntax-checked, but is treated as documentation for all items.

Format

```
[ { SYNCHRONIZED } [ LEFT  ] ]
[ { SYNC        } [ RIGHT ] ]
```

JUSTIFIED Clause

The JUSTIFIED clause overrides standard positioning rules for a receiving item of the alphabetic or alphanumeric categories.

Format

```
[ { JUSTIFIED } RIGHT ]
[ { JUST      }       ]
```

The JUSTIFIED clause can be specified only at the elementary level. JUST is an abbreviation for JUSTIFIED and has the same meaning.

The JUSTIFIED clause must not be specified for a numeric item or for any item for which editing is specified. The JUSTIFIED clause must not be specified with level-66 (RENAMES) or level-88 (condition-name) entries.

IBM Extension

The JUSTIFIED clause can be specified for an alphanumeric edited item.

End of IBM Extension

When the JUSTIFIED clause is specified for a receiving item, the data is aligned at the rightmost character position in the receiving item, and:

- If the sending item is larger than the receiving item, the leftmost characters are truncated.
- If the sending item is smaller than the receiving item, the unused character positions at the left are filled with spaces.

When the JUSTIFIED clause is omitted, the rules for standard alignment are followed.

The following shows the difference between standard and justified alignment when the receiving field has a length of five character positions:

Alignment	Sending Field Value	Receiving Field Value
Standard	THE	THE ␣ ␣
Justified right	THE	␣ ␣THE
Standard	TOO ␣ BIG	TOO ␣ B
Justified right	TOO ␣ BIG	O ␣ BIG

BLANK WHEN ZERO Clause

The BLANK WHEN ZERO clause specifies that an item is to be filled entirely with spaces when its value is zero. When the data item receives a value of zero through an explicit reference at run time, it is set to blanks.

Format

```
[ BLANK WHEN ZERO ]
```

The BLANK WHEN ZERO clause can be specified only for elementary numeric or numeric edited items. When it is specified for a numeric item, the item is considered to be a numeric edited item.

If the BLANK WHEN ZERO clause is specified, the item contains nothing but spaces when its value is zero.

The BLANK WHEN ZERO clause must not be specified for level-66 or level-88 items.

The BLANK WHEN ZERO clause and the asterisk (*) suppression symbol must not be specified for the same entry.

VALUE Clause

The VALUE clause specifies the initial contents of a data item, or the value(s) associated with a condition-name. The two formats for the VALUE clause are as follows:

Format 1

```
[ VALUE IS literal ]
```

Format 2

```
88 condition-name { VALUE IS   } literal-1 [ { THROUGH } literal-2 ]
                  { VALUES ARE }           [ { THRU    }         ]

[ literal-3 [ { THROUGH } literal-4 ] ] . . .   .
            [ { THRU    }           ]
```

Level-number 88 and condition-name are not part of the Format 2 VALUE clause itself, and are included in the format only for clarity. The use of the VALUE clause differs with the Data Division section in which it is specified.

File and Linkage Sections: The VALUE clause can only be used in condition-name entries.

Working-Storage Section: The VALUE clause is used in condition-name entries. It is also used to specify the initial value of any data item; the item assumes the specified value at the beginning of program processing. If the initial value is not explicitly specified, it is unpredictable.

General Considerations

The keywords THRU and THROUGH are equivalent.

The VALUE clause must not be specified for any item whose length is variable; that is, it is a group item that has an OCCURS DEPENDING ON clause subordinate to it.

For group entries, the VALUE clause must not be specified if the entry or an entry subordinate to it contains any of the following clauses: JUSTIFIED, SYNCHRONIZED, or USAGE (other than USAGE DISPLAY).

The VALUE clause must not conflict with other clauses in the data description entry or in the data description of this entry's hierarchy. The following rules apply:

- Wherever a literal is specified, a figurative constant can be substituted.
- If the item is numeric, all VALUE clause literals must be numeric literals. If the literal defines the value of a Working-Storage item, the literal is aligned according to the rules for numeric moves with one additional restriction: the literal must not have a value that requires truncation of nonzero digits. If the literal is signed, the associated PICTURE character-string must contain a sign symbol (S).
- All numeric literals in a VALUE clause of an item must have a value that is within the range of values indicated by the PICTURE clause for that item. For example, for PICTURE 99PPP, the literal must be zero or within the range 1000 through 99000. For PICTURE PPP99, the literal must be within the range .00000 through .00099.
- If the item is an alphabetic, alphanumeric, alphanumeric edited, or numeric edited item, all VALUE clause literals must be nonnumeric literals. The number of characters in the literal must not exceed the size of the item.

IBM Extension

- If the item is Boolean, the VALUE clause literal must be a Boolean literal.

End of IBM Extension

- The functions of the editing characters in a PICTURE clause are ignored in determining the initial appearance of the item described. However, editing characters are included in determining the size of the item. Therefore, any editing character must be included in the literal. For example, if the item is defined as PICTURE +999.99 and the value is to be +12.34, then the VALUE clause should be specified as VALUE "+012.34".
- A maximum of 32 767 bytes can be initialized by means of a single VALUE clause.

Format 1 Considerations

This format specifies the initial value of a data item in storage. Initialization is independent of any BLANK WHEN ZERO or JUSTIFIED clause specified.

A Format 1 VALUE clause must not be specified for an entry that contains or is subordinate to an entry that contains a REDEFINES or OCCURS clause.

If the VALUE clause is specified at the group level, the literal must be a nonnumeric literal or a figurative constant. The group area is initialized without consideration for the subordinate entries within this group. In addition, the VALUE clause must not be specified for subordinate entries within this group.

Format 2 Considerations

This format associates a value, values, and/or range(s) of values with a condition-name. Each such condition-name requires a separate level-88 entry.

The VALUE clause is required in a condition-name entry and must be the only clause in the entry. Each condition-name entry is associated with a preceding conditional variable. Thus, every level-88 entry must always be preceded either by the entry for the conditional variable or by another level-88 entry when several condition-names apply to one conditional variable. Such level-88 entries implicitly have the PICTURE characteristics of the conditional variable.

Every condition-name can be qualified by the name of its associated conditional variable and by the qualifier(s) of the conditional variable. If the associated conditional variable requires subscripts or indexes, each procedural reference to the condition-name must be subscripted or indexed as required for the conditional variable.

When only literal-1 is specified, the condition-name is associated with a single value.

When literal-1, literal-3 and so on are specified, the condition-name is associated with several single values.

When literal-1 THRU literal-2 is specified, the condition-name is associated with one range of values.

- If the literals are numeric, literal-1 must be less than literal-2.
- If the literals are nonnumeric, literal-1 must appear before literal-2 in the program collating sequence.

When literal-1 THRU literal-2, literal-3 THRU literal-4, and so on are specified, the condition-name is associated with more than one range of values.

- If the literals are numeric, literal-1 must be less than literal-2, literal-3 must be less than literal-4, and so on.
- If the literals are nonnumeric, literal-1 must appear before literal-2 in the program collating sequence, literal-3 must appear before literal-4 in the program collating sequence, and so on.

One or more single values and one or more ranges of values can be specified in a single Format 2 VALUE clause.

The type of literal in a condition-name entry must be consistent with the data type of the conditional variable. In the following example, CITY-COUNTY-INFO, COUNTY-NO, and CITY are conditional variables; the associated condition-names immediately follow the level-number 88. The PICTURE associated with COUNTY-NO limits the condition-name value to a two-digit numeric literal. The PICTURE associated with CITY limits the condition-name value to a three-character nonnumeric literal. Any values for the condition-names associated with CITY-COUNTY-INFO cannot exceed five characters, and the literal (because this is a group item) must be nonnumeric:

```
05 CITY-COUNTY-INFO.
      88 BRONX            VALUE "03NYC".
      88 BROOKLYN         VALUE "24NYC".
      88 MANHATTAN        VALUE "31NYC".
      88 QUEENS           VALUE "41NYC".
      88 STATEN-ISLAND    VALUE "43NYC".
   10 COUNTY-NO         PICTURE 99.
      88 DUTCHESS         VALUE 14.
      88 KINGS            VALUE 24.
      88 NEW-YORK         VALUE 31.
      88 RICHMOND         VALUE 43.
   10 CITY              PICTURE X(3).
      88 BUFFALO          VALUE "BUF".
      88 NEW-YORK-CITY    VALUE "NYC".
      88 POUGHKEEPSIE     VALUE "POK".
05 POPULATION...
```

The following example shows the use of the THRU phrase. In this example, the number of miles a person drives to work each day is categorized.

```
05 MILEAGE PIC 9(2)V9.
   88 LOW VALUE 0 THRU 09.9.
   88 MED VALUE 10.0 THRU 19.9.
   88 HIGH VALUE 20.0 THRU 99.9.
```

Condition-names are used procedurally in condition-name conditions, and are described under “Conditional Expressions” on page 354.

PICTURE Clause

The PICTURE clause specifies the general characteristics and editing requirements of an elementary item.

Format

```
[ { PICTURE } IS character-string ]
[ { PIC     }                     ]
```

The PICTURE clause must be specified for every elementary item except an indexed data item. The PICTURE clause can be specified only at the elementary level. PIC is an abbreviation for PICTURE and has the same meaning.

The character-string is made up of certain COBOL characters used as symbols. The allowable combinations determine the category of the data item. The character-string can contain a maximum of 30 characters.

Symbols Used in the PICTURE Clause

The functions of each PICTURE clause symbol are defined in the following list. Any punctuation character appearing within the PICTURE character-string is not considered a punctuation character, but rather as a PICTURE character-string symbol.

A Each A in the character-string represents a character position that can contain only a letter of the alphabet or a space.

B Each B in the character-string represents a character position into which the space character will be inserted.

P The P indicates an assumed decimal scaling position, and is used to specify the location of an assumed decimal point when the point is not within the number that appears in the data item. The scaling position character P is not counted in the size of the data item. Scaling position characters are counted in determining the maximum number of digit positions (18) in numeric edited items or in items that appear as arithmetic operands. In any operation converting data from one form of internal representation to another, if the item being converted is described with the PICTURE symbol P, each digit position described by a P is considered to contain the value zero, and the size of the item is considered to include these zero digit positions.

For example, PICTURE PPP99 DISPLAY defines a two-character item whose value is zero or ranges from .00001 through .00099. PICTURE 99PPP DISPLAY defines a two-character item whose value is zero or ranges from 1000 through 99000.

The scaling position character P can appear only to the left or right of the other characters in the string as a continuous string of Ps within a PICTURE description. The sign character S and the assumed decimal point V are the only characters which can appear to the left of a leftmost string of Ps. Because the scaling position character P implies an assumed decimal point (to the left of the Ps if the Ps are leftmost PICTURE characters; to the right of the Ps if the Ps are rightmost PICTURE characters), the assumed decimal point symbol V is redundant as either the leftmost or rightmost character within such a PICTURE description.

S The symbol S is used in a PICTURE character-string to indicate the presence (but not the representation or, necessarily, the position) of an operational sign. The sign must be written as the leftmost character in the PICTURE string. An operational sign indicates whether the value of an item involved in an operation is positive or negative. The symbol S is not counted in determining the size of the elementary item, unless an associated SIGN clause specifies the SEPARATE CHARACTER option.

V The V is used in a character-string to indicate the location of the assumed decimal point and can appear only once in a character-string. The V does not represent a character position and, therefore, is not counted in the size of the elementary item. When the assumed decimal point is to the right of the rightmost symbol in the string, the V should not be included in this PICTURE string.

X Each X in the character-string represents a character position that can contain any allowable character from the EBCDIC set.

Z Each Z in the character-string represents a leading numeric character position. When that position contains a zero, the zero is replaced by a space character. Each Z is counted in the size of the item.

9 Each 9 in the character-string represents a character position that contains a numeral and is counted in the size of the item.

IBM Extension

1 The character 1 represents a character position that contains a Boolean value of B"1" or B"0". Usage must be explicitly or implicitly defined as DISPLAY.

End of IBM Extension

0 Each zero in the character-string represents a character position into which the numeral zero will be inserted. Each zero is counted in the size of the item.

/ Each slash in the character-string represents a character position into which the slash character will be inserted. Each slash is counted in the size of the item.

, Each comma in the character-string represents a character position into which a comma will be inserted. This character is counted in the size of the item. The comma insertion character cannot be the last character in the PICTURE character-string.

. When a period appears in the character-string, it is an editing symbol that represents the decimal point for alignment purposes. In addition, it represents a character position into which a period will be inserted. This character is counted in the size of the item. The period insertion character cannot be the last character in the PICTURE character-string.

Note: For a given program, the functions of the period and comma are exchanged if the clause DECIMAL-POINT IS COMMA is stated in the SPECIAL-NAMES paragraph. In this exchange, the rules for the period apply to the comma, and the rules for the comma apply to the period wherever they appear in a PICTURE clause.

`+, –, CR, DB`

These symbols are used as editing sign control symbols. Each symbol represents the character position into which the editing sign control symbol will be placed. The symbols are mutually exclusive in one character-string. Each character used in the symbol is counted in determining the size of the data item.

* Each asterisk (check protect symbol) in the character-string represents a leading numeric character position into which an asterisk will be placed when that position contains a zero. Each asterisk is counted in the size of the item.

The asterisk used as the zero suppression symbol and the `BLANK WHEN ZERO` clause must not appear in the same entry.

'cs' The currency symbol in the character-string represents a character position into which a currency symbol is to be placed. The currency symbol in a character-string is represented either by the symbol `$` or by the single character specified in the `CURRENCY SIGN` clause in the `SPECIAL-NAMES` paragraph of the Environment Division. The currency symbol is counted in the size of the item.

Note: Because the currency symbol can be replaced in the `CURRENCY SIGN` clause, the term **'cs'** is used throughout this book rather than the actual currency symbol ($).

Figure 73 on page 335 gives the order in which `PICTURE` clause symbols must be specified.

Character-String Representation: The following symbols can appear more than once in one `PICTURE` character-string:

`A B P X Z 9 0 / , + – * 'cs'`

Each time one of these symbols appears in the character-string, it represents an occurrence of that character or set of allowable characters in the data item.

First Symbol / Second Symbol		Nonfloating Insertion Symbols									Floating Insertion Symbols						Other Symbols						
		B	0	/	,	.	{+ -}[1]	{+}[1]	{CR DB}	'cs'[2]	{Z *}[1]	{Z *}[1]	{+}[1]	{-}[1]	'cs'[2]	'cs'[2]	9	A X	S	V	P[1]	P[1]	1
Nonfloating Insertion Symbols	B	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	0	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	/	x	x	x	x	x	x			x	x	x	x	x	x	x	x	x		x		x	
	,	x	x	x	x	x	x			x	x	x	x	x	x	x	x			x		x	
	.	x	x	x	x		x			x	x		x		x		x						
	{+ -}																						
	{+ -}	x	x	x	x	x				x	x	x			x	x	x			x	x	x	
	{CR DB}	x	x	x	x	x				x	x	x			x	x	x			x	x	x	
	'cs'						x																
Floating Insertion Symbols	{Z *}	x	x	x	x		x			x	x												
	{Z *}	x	x	x	x	x	x			x	x	x								x		x	
	{+ -}	x	x	x	x					x			x										
	{+ -}	x	x	x	x	x				x			x	x						x		x	
	'cs'	x	x	x	x		x								x								
	P	x	x	x	x	x	x								x	x				x		x	
Other Symbols	9	x	x	x	x	x	x			x	x		x		x		x	x	x	x		x	
	A X	x	x	x													x	x					
	S																						
	V	x	x	x	x		x			x	x		x		x		x		x		x		
	P	x	x	x	x		x			x	x		x		x		x		x		x		
	P						x			x									x	x		x	
	1																						

[1] Nonfloating insertion symbols + and -, floating insertion symbols Z, *, +, -, and 'cs', and other symbol P appear twice in the above table. The leftmost column and uppermost row for each symbol represents its use to the left of the decimal point position. The second appearance of the symbol in the table represents its use to the right of the decimal point position.

[2] $ is the default value for the currency symbol. This value can be replaced by a character specified in the currency SIGN clause.

An X at an intersection indicates that the symbol(s) at the top of the column can, in a given character string, appear anywhere to the left of the symbol(s) at the left of the row.

Braces ({ }) indicate items that are mutually exclusive.

At least one of the symbols A, X, Z, 9, or *, or at least two of the symbols +, -, or 'cs' must be present in a PICTURE string.

Figure 73. PICTURE Clause Symbol Order

Note: The numeral ‘1’ in the Other Symbols categories represents an IBM Extension to ANS COBOL X3.23-1974.

An integer enclosed in parentheses immediately following any of these symbols specifies the number of consecutive occurrences of that symbol. The number of consecutive occurrences cannot exceed 32 767.

For example, the following two PICTURE clause specifications are equivalent:

```
PICTURE IS $99999.99CR
PICTURE IS $9(5).99CR
```

The following five symbols can each appear only once in one PICTURE character-string:

```
S V . CR DB 1
```

Data Categories and PICTURE Considerations: The allowable combinations of PICTURE symbols determine the data category of the item. Rules for each category follow.

The following rules apply for alphabetic items:

- The PICTURE character-string can contain only the symbols A and B.
- The contents of the item in standard data format must consist of any of the 26 letters of the alphabet and the space character.
- USAGE DISPLAY must be either specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal.

IBM Extension

Boolean items–the following rules apply:

- The PICTURE character-string can contain only the symbol 1.
- Only one character 1 can be specified.
- The USAGE of an item can only be DISPLAY.
- An associated VALUE clause must specify a Boolean literal (B"1" or B"0") or zero.
- The following clauses cannot be specified for a Boolean item:
  - SIGN clause
  - BLANK WHEN ZERO clause
  - ASCENDING/DESCENDING KEY clause.
- The INDICATOR clause can be specified (see “Indicators” on page 92).

End of IBM Extension

The following rules apply for numeric items:

- The PICTURE character-string can contain only the symbols 9, P, S, and V.
- The number of digit positions must range from 1 through 18.
- The contents of a numeric item must be a combination of the digits 0 through 9. The numeric item can contain an operational sign. If the PICTURE contains an S, the contents of the item are treated as a positive or negative value, depending on the operational sign present in the data. If the PICTURE does not contain an S, the contents of the item are treated as an absolute value.
- If a VALUE clause is specified for an elementary numeric item, the literal must be numeric. If a VALUE clause is specified for a group item consisting of elemen-

tary numeric items, the group is considered alphanumeric, and the literal must therefore be nonnumeric.

- The USAGE clause of the item can be DISPLAY or COMPUTATIONAL.

IBM Extension

The USAGE can be COMPUTATIONAL-3 or COMPUTATIONAL-4.

End of IBM Extension

Examples of numeric items are shown in Table 7.

Table 7. Examples of Numeric Items

PICTURE	Valid Range of Values
9999	0 through 9999
S99	–99 through +99
S999V9	–999.9 through +999.9
PPP999	0 through .000999
S999PPP	–1000 through –999000 and +1000 through +999000 or zero

The following rules apply to alphanumeric items:

- The PICTURE character-string must consist of either:
  - The symbol X entirely.
  - Combinations of the symbols A, X, and 9. The item is treated as if the character-string contained only the symbol X. A PICTURE character-string containing all A's or all 9's does not define an alphanumeric item.
- The contents of the item in standard data format may be any allowable characters from the EBCDIC character set.
- USAGE DISPLAY must be either specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal.

The following rules apply to alphanumeric edited items:

- The PICTURE character-string can contain the symbols:

  A X 9 B 0 /

- The string must contain at least one of the following combinations:
  - At least one B and at least one X
  - At least one 0 and at least one X
  - At least one X and at least one /
  - At least one A and at least one 0
  - At least one A and at least one /.
- The contents of the item in standard data format can be any allowable character from the EBCDIC character set.
- USAGE DISPLAY must be either specified or implied.

- Any associated VALUE clause must specify a nonnumeric literal. The literal is treated exactly as specified; no editing is performed.
- Alphanumeric edited items are subject to only one type of editing–simple insertion using the symbols 0, B, and /.

The following rules apply to numeric edited items:

- The PICTURE character-string can contain the following symbols:

  B P V Z 9 0 / , . + – CR DB * 'cs'

  The combinations of symbols allowed are determined from the PICTURE clause symbol order allowed (see Figure 73 on page 335), and the editing rules (see the following section). The following additional rules also apply:

  – The string must contain at least one of the following symbols:

    B / Z 0 , . * + – CR DB 'cs'

  – The number of digit positions represented in the character-string must be in the range of 1 through 18 inclusive.
  – The total number of character positions in the string (including editing characters) must not exceed 30.
- The contents of those character positions representing digits in standard data format must be one of the digits 0 through 9.
- USAGE DISPLAY must be either specified or implied.
- Any associated VALUE clause must specify a nonnumeric literal. The literal is treated exactly as specified; no editing is performed.

PICTURE Clause Editing

There are two general methods of processing editing in a PICTURE clause: by insertion, or by suppression and replacement.

There are four types of insertion editing: simple insertion, special insertion, fixed insertion, and floating insertion. There are two types of suppression and replacement editing: zero suppression and replacement with asterisks, and zero suppression and replacement with spaces.

The type of editing allowed for an item depends on its data category. The type of editing that is valid for each category is shown in Table 8.

Each type of editing is discussed in detail in the following paragraphs.

Simple Insertion Editing: This type of editing is valid for alphabetic, alphanumeric edited, and numeric edited items. The valid insertion symbols for each category are shown in Table 9 on page 339.

Each insertion symbol is counted in the size of the item, and represents the position within the item where the equivalent characters will be inserted. Examples of simple insertion editing are shown in Table 10 on page 339.

Table 8 (Page 1 of 2). Valid Editing for Each Data Category

Category	Type of Editing
Alphabetic	Simple insertion

Table 8 (Page 2 of 2). Valid Editing for Each Data Category	
Category	**Type of Editing**
Boolean	None
Numeric	None
Alphanumeric	None
Alphanumeric edited	Simple insertion
Numeric edited	All

Table 9. Valid Insertion Symbols for Simple Insertion Editing for Each Data Category	
Category	**Valid Insertion Symbols**
Alphabetic	B
Boolean	None
Numeric	None
Alphanumeric	None
Alphanumeric edited	B 0 /
Numeric edited	B 0 / ,

Table 10. Examples of Simple Insertion Editing		
PICTURE	**Value of Data**	**Edited Result**
X(10)/XX	ALPHANUMER01	ALPHANUMER/01
X(5)BX(7)	ALPHANUMERIC	ALPHA NUMERIC
A(5)BA(5)	ALPHABETIC	ALPHA BETIC
99,B999,B000	1234	01, 234, 000
99,999	12345	12,345

Special Insertion Editing: This type of editing is valid only for numeric edited items.

The period is the special insertion symbol; it also represents the actual decimal point for alignment purposes.

The period insertion symbol is counted in the size of the item, and represents the position within the item where the actual decimal point will be inserted.

The actual decimal point and the assumed decimal point (the symbol V) must not both be specified in one PICTURE character-string.

Examples of special insertion editing are shown in Table 11 on page 340.

Fixed Insertion Editing: This type of editing is valid only for numeric edited items. The following insertion symbols are used:

- 'cs' (currency symbol)
- + – CR DB (editing sign control symbols).

- In fixed insertion editing, only one currency symbol and one editing sign control symbol can be specified in one PICTURE character-string.
- Unless it is preceded by a + or – symbol, the currency symbol must be the leftmost character position in the character-string.
- When either + or – is used as a symbol, it must represent either the leftmost or rightmost character position in the character-string.

Table 11. Examples of Special Insertion Editing

PICTURE	Value of Data	Edited Results
999.99	1.234	001.23
999.99	12.34	012.34
999.99	123.45	123.45
999.99	1234.5	234.50

- When CR or DB is used as a symbol, it must represent the rightmost two character positions in the character-string.
- Editing sign control symbols produce results depending on the value of the data item as shown in Table 12.

Examples of fixed insertion editing are shown in Table 13.

Table 12. Editing Sign Control Results

Editing Symbol in PICTURE Character String	Resulting Data Item Positive or Zero	Resulting Data Item Negative
+	+	–
–	space	–
CR	2 spaces	CR
DB	2 spaces	DB

Table 13. Examples of Fixed Insertion Editing

PICTURE	Value of Data	Edited Result
999.99+	+6555.556	555.55+
+9999.99	–6555.555	–6555.55
9999.99	+1234.56	1234.56
$999.99	–123.45	$123.45
–$999.99	–123.456	–$123.45
$9999.99CR	+123.45	$0123.45
$9999.99DB	–123.45	$0123.45DB

Floating Insertion Editing: This type of editing is valid only for numeric edited items. The following symbols are used:

'cs' + –

Within one PICTURE character-string, these symbols are mutually exclusive as floating insertion characters.

Floating insertion editing is specified by using a string of at least two of the allowable floating insertion symbols to represent leftmost character positions in which these characters can be inserted.

The leftmost floating insertion symbol in the character-string represents the leftmost limit at which this character can appear in the data item. The rightmost floating insertion symbol represents the rightmost limit at which this character can appear.

The second leftmost floating insertion symbol in the character-string represents the leftmost limit at which numeric data can appear within the data item. Nonzero numeric data can replace all characters at or to the right of this limit.

Any simple insertion symbols (B 0 / ,) within or to the immediate right of the string of floating insertion symbols are considered part of the floating character-string.

In a PICTURE character-string there are two methods to represent floating insertion editing and to process editing:

- Any or all leading numeric character positions to the left of the decimal point are represented by the floating insertion symbol. When editing is processed, a single floating insertion character is placed to the immediate left of the first nonzero digit in the data or of the decimal point, whichever is farther left. The character positions to the left of the inserted character are filled with spaces.
- All the numeric character positions are represented by the floating insertion symbol. When editing is processed, then:
  - If the value of the data is zero, the entire data item will contain spaces.
  - If the value of the data is nonzero, the result is the same as in method 1.

To avoid truncation, the minimum size of the PICTURE character-string must be the sum of:

- The number of character positions in the sending item
- The number of nonfloating insertion symbols in the receiving item
- One character for the floating insertion symbol.

Examples of floating insertion editing are shown in Table 14.

Table 14. Examples of Floating Insertion Editing

PICTURE	Value of Data	Edited Result
$$$$.99	.123	$.12
$$$9.99	.12	$0.12
$$,$$$,999.99	–1234.56	$1,234.56
++,+++,999.99	–123456.789	–123,456.78
$$,$$$,$$$.99CR	–1234567	$1,234,567.00CR
++,+++,+++,+++.++	0000.00	

Zero Suppression and Replacement Editing: This type of editing is valid only for numeric edited items.

The symbols `Z` and `*` are used for zero suppression. These symbols are mutually exclusive in the `PICTURE` clause.

The following symbols are mutually exclusive as floating replacement symbols in one `PICTURE` character-string:

```
Z * + - 'cs'
```

Zero suppression editing is specified by using a string of one or more of the allowable symbols to represent leftmost character positions in which zero suppression and replacement editing can be processed.

Any simple insertion symbols (`B 0 / ,`) within or to the immediate right of the string of floating editing symbols are considered part of the string.

In a `PICTURE` character-string, there are two ways to represent zero suppression and process editing:

- Any or all of the leading numeric character positions to the left of the decimal point are represented by suppression symbols. When editing is processed, any leading zero in the data that appears in the same character position as a suppression symbol is replaced by the replacement character. Suppression stops at the character farthest left that:
  - Does not correspond to a suppression symbol.
  - Contains nonzero data.
  - Is the decimal point.
- All the numeric character positions in the `PICTURE` character-string are represented by the suppression symbols. When editing is processed and the value of the data is nonzero, the result is the same as in the preceding rule. The following rules apply if the value of the data is zero:
  - If `Z` has been specified, the entire data item contains spaces.
  - If `*` has been specified, the entire data item, except the actual decimal point, contains asterisks.

The asterisk as a suppression symbol and the `BLANK WHEN ZERO` clause must not be specified for the same entry.

Examples of zero suppression and replacement editing are shown in Table 15.

Table 15 (Page 1 of 2). Examples of Zero Suppression and Replacement Editing

PICTURE	Value of Data	Edited Result
****.**	0000.00	****.**
ZZZZ.ZZ	0000.00	
ZZZZ.99	0000.00	.00
****.99	0000.00	****.00
ZZ99.99	0000.00	00.00
Z,ZZZ.ZZ+	+123.456	123.45+

Table 15 (Page 2 of 2). Examples of Zero Suppression and Replacement Editing

PICTURE	Value of Data	Edited Result
*,***.**+	–123.45	**123.45
,*,***.**+	+12345678.9	12,345,678.90+
$Z,ZZZ,ZZZ.ZZCR	+12345.67	$ 12,345.67
$B*,***,***.**BDB	–12345.67	$ ***12,345.67 DB

RENAMES Clause

The RENAMES clause specifies alternative, possibly overlapping, groupings of elementary data items. This clause allows a single data-item to rename a group of data items within a record.

Format

```
66 data-name-1 RENAMES data-name-2 [ { THROUGH } data-name-3 ] .
                                   [ { THRU    }             ]
```

Note: Level-number 66 and data-name-1 are not part of the RENAMES clause itself, and are included in the format only for clarity.

One or more RENAMES entries can be written for a logical record. All RENAMES entries associated with one logical record must immediately follow that record's last data description entry.

Data-name-1 identifies an alternative grouping of data items. It cannot be used as a qualifier; it can be qualified only by the names of FD or SD entries or level-01 entries.

A level-66 entry cannot rename a level-01, level-77, level-88, or another level-66 entry.

Data-name-2 or data-name-3 identifies the original grouping of elementary data items; that is, they must name elementary or group items within the associated level-01 entry and must not be the same data-name. Both data-names may be qualified.

The OCCURS clause must not be specified in the data entries for data-name-2 and data-name-3, or for any group entry to which they are subordinate. In addition, the OCCURS DEPENDING ON clause must not be specified for any item occupying storage between data-name-2 and data-name-3.

When data-name-2 is specified, and data-name-3 is not specified, data-name-1 is defined with the same attributes as data-name-2.

When both data-name-2 and data-name-3 are specified, the following occurs:

- If data-name-2 is an elementary item, data-name-1 is defined as a group item starting with data-name-2 and ending with data-name-3, or the last elementary item in data-name-3, if it is a group item.

- If data-name-2 is a group item, data-name-1 is defined as a group item starting with the first elementary item in data-name-2, and ending with data-name-3, or the last elementary item in data-name-3, if it is a group item.

The keywords THRU and THROUGH are equivalent.

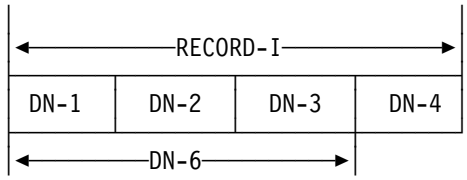
The leftmost character in data-name-3 must not precede that in data-name-2; the rightmost character in data-name-3 must follow that in data-name-2. This means that data-name-3 cannot be subordinate to data-name-2.

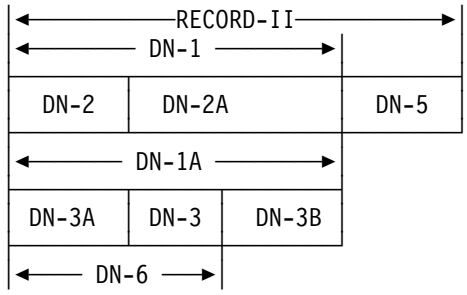
Valid and invalid specifications of the RENAMES clause are given in Figure 74 on page 345.

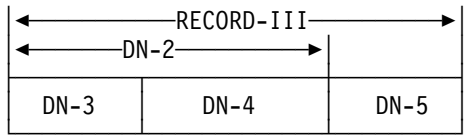
COBOL Specifications

```
EXAMPLE 1 (Valid)

   01  RECORD-I.
       05  DN-1...  .
       05  DN-2...  .
       05  DN-3...  .
       05  DN-4...  .
   66  DN-6 RENAMES DN-1 THROUGH DN-3.

EXAMPLE 2 (Valid)

   01  RECORD-II.
       05  DN-1.
           10  DN-2...  .
           10  DN-2A...  .
       05  DN-1A REDEFINES DN-1.
           10  DN-3A...  .
           10  DN-3...  .
           10  DN-3B...  .
       05  DN-5... .
   66  DN-6 RENAMES DN-2 THROUGH DN-3.

EXAMPLE 3 (Invalid)

   01  RECORD-III
       05  DN-2.
           10  DN-3...  .
           10  DN-4...  .
       05  DN-5...  .
   66  DN-6 RENAMES DN-2 THROUGH DN-3.

EXAMPLE 4 (Invalid)

   01  RECORD-IV.
       05  DN-1.
           10  DN-2A...  .
           10  DN-2B...  .
           10  DN-2C REDEFINES DN-2B.
               15  DN-2...  .


               15  DN-2D... .
       05  DN-3... .

   66  DN-4 RENAMES DN-1 THROUGH DN-2.
```

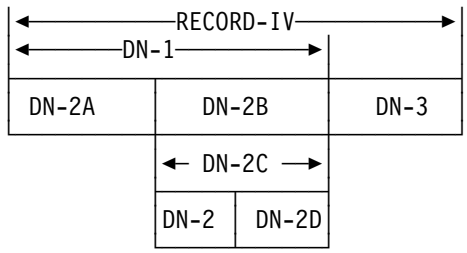
Storage Layouts







DN-6 is indeterminate.



DN-4 is indeterminate

Figure 74. Valid and Invalid Specifications of the RENAMES Clause

Chapter 10. Procedure Division

Procedure Division Concepts

The Procedure Division is required in every COBOL source program. The Procedure Division consists of optional Declaratives and procedures that contain the sections and/or paragraphs, sentences, and statements that solve a data processing problem.

Declaratives

The Declarative section provides a method of calling procedures that are run when an exceptional condition occurs that is to be tested by the COBOL programmer.

When Declarative sections are specified, they must be grouped at the beginning of the Procedure Division. Declarative sections are preceded by the keyword DECLARATIVES and followed by the keywords END DECLARATIVES.

If Declarative sections are specified, the entire Procedure Division must be divided into sections.

Procedures

A *procedure* is a paragraph, group of paragraphs, a section, or a group of sections within the Procedure Division. A *procedure-name* is a user-defined name that identifies a section or a paragraph.

A *section* consists of a section header followed by zero, one, or more than one successive paragraphs. A *section-header* is a section-name followed by the keyword SECTION, an optional segment-number, followed by a period and a space. Segment-numbers are explained under “Segmentation Feature” in Chapter 11, “Using the Additional COBOL Functions.” A *section-name* is a user-defined word that identifies a section. A section-name, because it cannot be qualified, must be unique. A section ends immediately before the next section header, at the end of the Procedure Division, or, in the Declaratives portion, at the keywords END DECLARATIVES.

A *paragraph* consists of a paragraph-name followed by a period and a space. Zero, one, or more than one successive sentences are allowed. A *paragraph-name* is a user-defined word that identifies a paragraph. A paragraph-name, because it can be qualified, need not be unique. A paragraph ends immediately before the next paragraph-name or section header, at the end of the Procedure Division, or, in the Declaratives portion, at the keywords END DECLARATIVES. If one paragraph in a program is contained within a section, then all paragraphs must be contained in sections.

A *sentence* consists of one or more statements terminated by a period and a space.

A *statement* is a syntactically valid combination of words (identifiers, data-names, figurative constants, and so on) and symbols (literals, relational-operators, and so on) beginning with a COBOL verb.

A *data-name* is a user-defined word naming a data item described in a data description entry in the Data Division. When *data-name* is used in a general format, it represents a word that cannot be subscripted, indexed, or qualified unless this is specifically permitted by the rules for that format.

An *identifier* consists of the word or words necessary to make unique reference to a data item through qualification, subscripting, or indexing. In any Procedure Division reference except the class test, if the contents of an identifier are not compatible with the class specified through its PICTURE clause, results are unpredictable.

Note: A level-88 (condition-name) entry, because it is not a data item, cannot be an identifier. The associated conditional variable, however, can be an identifier.

Procedure Division Organization

The structure of the Procedure Division is shown in the following formats. Figure 75 on page 349 gives an example.

Format 1

```
  PROCEDURE DIVISION [ USING data-name-1 [ , data-name-2 ] . . . ] .

[ DECLARATIVES.

{ section-name SECTION [ segment-number ] . use-sentence.
[ paragraph-name. [ sentence ] ... ] . . . } . . .

  END DECLARATIVES. ]

{ section-name SECTION [ segment-number ] .
[ paragraph-name. [ sentence ] ... ] . . . } . . .
```

Format 2

```
  PROCEDURE DIVISION [ USING data-name-1 [ , data-name-2 ] . . . ] .

{ paragraph-name. [ sentence ] . . . } . . .
```

Coding Example

```
SEQUENCE      C  A       B
              O
PAGE  SERIAL  N
              T
1   3 4   6   7 8       12    16    20    24    28    32    36
004010      PROCEDURE DIVISION.
   |  020      DECLARATIVES.
   |  030      SECTION-NAME SECTION.
   |  040      PARAGRAPH-NAMES.
   |  050        PROGRAMMING STATEMENTS.
   |  060*       COMMENTS.
   |  070    END DECLARATIVES.
   |  080      SECTION-NAME SECTION.
   |  090      PARAGRAPH-NAME.
   ▼  100        PROGRAMMING STATEMENTS.
```

Figure 75. Coding Example to Show Procedure Division Organization

Categories of Sentences

There are three categories of sentences: conditional sentences, imperative sentences, and compiler-directing sentences.

A *conditional sentence* is a conditional statement, optionally preceded by an imperative statement, terminated by a period and a space.

An *imperative sentence* is an imperative statement, which can consist of a series of imperative statements, followed by a period and a space.

A *compiler-directing sentence* is a single compiler-directing statement, followed by a period and a space.

Categories of Statements

Three categories of statements are used in COBOL: conditional statements, imperative statements, and compiler-directing statements.

A *conditional statement* specifies that the truth value of a condition is to be determined, and that the subsequent action of the object program is dependent on this truth value. Table 16 on page 350 lists types of COBOL conditional statements.

An *imperative statement* specifies that an unconditional action is to be taken by the object program. An imperative statement can also consist of a series of imperative statements. Table 17 on page 350 lists types of COBOL imperative statements.

A *compiler-directing statement* causes the compiler to take a specific action during compilation. Table 18 on page 351 lists types of COBOL compiler-directing statements.

Table 16. Types of Conditional Statements

Type	Conditional Statement
Decision	IF
Input/Output	DELETE...INVALID KEY READ...AT END READ...INVALID KEY REWRITE...INVALID KEY START...INVALID KEY WRITE...AT END-OF-PAGE WRITE...INVALID KEY
Arithmetic	ADD...ON SIZE ERROR COMPUTE...ON SIZE ERROR DIVIDE...ON SIZE ERROR MULTIPLY...ON SIZE ERROR SUBTRACT...ON SIZE ERROR
Data Movement	STRING...ON OVERFLOW UNSTRING...ON OVERFLOW
Table Handling	SEARCH
Ordering	RETURN...AT END
Inter-program Communication	CALL...ON OVERFLOW
Procedure Branching	PERFORM...UNTIL

Table 17 (Page 1 of 2). Types of Imperative Statements

Type	Imperative Statement
Arithmetic	ADD[9] COMPUTE[9] DIVIDE[9] INSPECT (TALLYING) MULTIPLY[9] SUBTRACT[9]
Data Movement	ACCEPT (DATE, DAY, TIME) INSPECT (REPLACING) MOVE STRING[11] UNSTRING[11]
Ending	STOP RUN EXIT PROGRAM
Input/Output	ACCEPT ACQUIRE CLOSE COMMIT DELETE[10] DISPLAY DROP OPEN READ[12] REWRITE[10] ROLLBACK START[10] STOP literal WRITE[13]
Ordering	MERGE RELEASE SORT
Procedure Branching	ALTER EXIT GO PERFORM[14]

Table 17 (Page 2 of 2). Types of Imperative Statements	
Type	**Imperative Statement**
Table Handling	SET
Inter-program Communication	CALL[11] CANCEL

Table 18. Types of Compiler-Directing Statements	
Type	**Compiler-Directing Statement**
Library	COPY
Declarative	USE
Documentation	ENTER

Categories of Expressions

Two categories of expressions are used in COBOL: arithmetic expressions and conditional expressions.

Arithmetic expressions are used as operands of conditional or arithmetic statements.

Conditional expressions cause the object program to select alternative paths of control, depending on the value of a truth test. There are two types of conditional expressions: simple conditions and complex conditions. Conditional expressions can be specified in the IF, PERFORM, and SEARCH statements.

9 Without the SIZE ERROR phrase.

10 Without the INVALID KEY phrase.

11 Without the ON OVERFLOW phrase.

12 Without the AT END, INVALID KEY, or NO DATA phrase.

13 Without the INVALID KEY or END-OF-PAGE phrase.

14 Without the UNTIL phrase.

Example Procedure Division Statements

```
. 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7

PROCEDURE DIVISION.
DECLARATIVES.
ERROR-IT SECTION.
    USE AFTER STANDARD ERROR PROCEDURE ON INPUT-DATA.
ERROR-ROUTINE.
    IF CHECK-IT = "30" ADD 1 TO DECLARATIVE-ERRORS.
END DECLARATIVES.
BEGIN-NON-DECLARATIVES SECTION.
100-BEGIN-IT.
    OPEN INPUT INPUT-DATA OUTPUT REPORT-OUT.
110-READ-IT.
    READ INPUT-DATA RECORD
       AT END MOVE "Y" TO EOF-SW.
    IF EOF-SW NOT = "Y" ADD 1 TO RECORDS-IN.
200-MAIN-ROUTINE.
    PERFORM PROCESS-DATA UNTIL EOF-SW = "Y".
    PERFORM FINAL-REPORT THRU FINAL-REPORT-EXIT.
    DISPLAY "TOTAL RECORDS IN = " RECORDS-IN UPON WORK-STATION.
    DISPLAY "DECLARATIVE ERRORS = " DECLARATIVE-ERRORS
       UPON WORK-STATION.
    STOP RUN.
PROCESS-DATA.
    IF RECORD-ID = "G"
        PERFORM PROCESS-GEN-INFO
    ELSE
        IF RECORD-CODE = "C"
           PERFORM PROCESS-SALES-DATA
        ELSE
           PERFORM UNKNOWN-RECORD-TYPE.
```

Arithmetic Expressions

Arithmetic expressions are used as operands of certain conditional and arithmetic statements. An arithmetic expression can consist of any of the following:

- An identifier described as a numeric elementary item
- A numeric literal
- Identifiers and literals, as defined in Items 1 and 2, separated by arithmetic operators
- Two arithmetic expressions, as defined in Items 1, 2, and/or 3, separated by an arithmetic operator
- An arithmetic expression, as defined in Items 1, 2, 3, and/or 4, enclosed in parentheses.

Any arithmetic expression can be preceded by a unary operator.

Identifiers and literals appearing in an arithmetic expression must represent either numeric elementary items or numeric literals on which arithmetic can be processed.

Arithmetic Operators

The five binary arithmetic operators and two unary arithmetic operators shown in Table 19 can be used in arithmetic expressions. The arithmetic operators are represented by specific characters that must be preceded and followed by a space.

Parentheses can be used in arithmetic expressions to specify the order in which elements are to be evaluated. Expressions within parentheses are evaluated first. When expressions are contained within a nest of parentheses, evaluation proceeds from the innermost to the outermost set of parenthetical expressions.

When parentheses are not used, or when parenthesized expressions are at the same level of inclusiveness, the following hierarchical order is implied:

1. Unary operator
2. Exponentiation
3. Multiplication and division
4. Addition and subtraction.

Parentheses either eliminate ambiguities in logic where consecutive operations appear at the same hierarchical level or modify the normal hierarchical sequence of processing when this is necessary. When the order of consecutive operations at the same hierarchical level is not completely specified by parentheses, the order is from left to right.

Figure 76 on page 354 shows permissible arithmetic symbol pairs. An arithmetic symbol pair is the appearance of two such symbols in sequence.

An arithmetic expression can begin only with a left parenthesis, a unary operator, or a variable (that is, an identifier or literal). An arithmetic expression can end only with a right parenthesis or a variable. An arithmetic expression must contain at least one reference to an identifier or literal. There must be a one-to-one correspondence between left and right parentheses in an arithmetic expression; each left parenthesis is placed to the left of its corresponding right parenthesis.

Note: The results of exponentiation are truncated after the thirteenth fractional digit. The results of exponentiation when the exponent is noninteger are accurate to seven digits.

Table 19. Binary and Unary Operators

Binary Operator	**Meaning**
+	Addition
–	Subtraction
*	Multiplication
/	Division
**	Exponentiation
Unary Operator	**Meaning**
+	Multiplication by +1; retains original sign
–	Multiplication by –1; changes sign

First Symbol	Second Symbol				
	Variable (identifier or literal)	* / ** + -	Unary + Unary -	(	)
Variable (identifier or literal)	-	p	-	-	p
* / ** + -	p	-	p	p	-
Unary + or Unary -	p	-	-	p	-
(	p	-	p	p	-
)	-	p	-	-	p

p indicates a permissible pairing
- indicates that the pairing is not permitted

Figure 76. Valid Arithmetic Symbol Pairs

Conditional Expressions

A conditional expression causes the object program to select alternative paths of control, depending on the truth value of a test. Conditional expressions can be specified in IF, PERFORM, and SEARCH statements. A conditional expression can be specified in simple conditions and in complex conditions. Both simple and complex conditions can be enclosed within any number of paired parentheses; parentheses do not change the category of the condition.

Simple Conditions

There are five simple conditions: class condition, condition-name condition, relation condition, sign condition, and switch-status condition. A simple condition has a truth value of true or false. When a simple condition is enclosed in paired parentheses, its truth value is not changed.

Class Condition

The class condition determines whether a data item is alphabetic or numeric.

Format

```
identifier IS [ NOT ] { NUMERIC    }
                      { ALPHABETIC }
```

The identifier being tested must be described implicitly or explicitly as USAGE DISPLAY. This identifier is determined to be numeric only if the contents consist of any combination of the digits 0 through 9.

If the PICTURE of the identifier being tested does not contain an operational sign, the identifier is determined to be numeric only if the contents are numeric and an operational sign is not present.

If the PICTURE of the identifier being tested does contain an operational sign, the identifier is determined to be numeric only if the item is an elementary item, the contents are numeric, and a valid operational sign is present.

In the EBCDIC collating sequence, valid embedded operational positive signs are hexadecimal F, C, E, and A. Negative signs are hexadecimal D and B. The preferred positive sign is hexadecimal F, and the preferred negative sign is hexadecimal D. For items described with the SIGN IS SEPARATE clause, valid operational signs are + (hex 4E) and - (hex 60).

The NUMERIC test cannot be used with an identifier described either as alphabetic or as a group item that contains one or more signed elementary items. The identifier being tested is determined to be alphabetic only if the contents consist of any combination of the alphabetic characters A through Z and the space.

The ALPHABETIC test cannot be used with an identifier described as numeric.

Table 20 shows valid forms of the class test.

Table 20. Valid Forms of the Class Test

Type of Identifier	Valid Forms of the Class Test
Alphabetic	ALPHABETIC NOT ALPHABETIC
Alphanumeric, alphanumeric edited, or numeric edited	ALPHABETIC NOT ALPHABETIC NUMERIC NOT NUMERIC
Zoned decimal	NUMERIC NOT NUMERIC

Condition-Name Condition

A condition-name condition causes a conditional variable to be tested to determine whether its value is equal to any of the values associated with the condition-name (level-88 item).

Format

```
condition-name
```

A condition-name is used in conditions as an abbreviation for the relation condition, because the specified condition-name is equal to only one of the values or ranges of values assigned to the specified conditional variable. The result of the test is true if one of the values corresponding to the condition-name equals the current value of the associated conditional variable.

If the condition-name is associated with a range of values or with several ranges of values, the conditional variable is tested to determine whether or not its value falls within the range(s), including the end values. The result of the test is true if one of the values corresponding to the condition-name equals the value of its associated conditional variable.

The following example illustrates the usage of condition-names and conditional variables:

```
02 GRADE-ID PIC 99.
   88 PRIMARY-OTHER
       VALUE 1 THRU 3, 5, 6.
   88 PRIMARY-FOUR
       VALUE 4.
   88 JUNIOR-HI
       VALUE 7 THRU 9.
   88 SENIOR-HI
       VALUE 10 THRU 12.
```

GRADE-ID is the conditional variable, PRIMARY-OTHER, PRIMARY-FOUR, JUNIOR-HI, and SENIOR-HI are condition-names. For individual records in the file, only one of the values specified in the condition-name entries can be present. To determine the grade level of a specific record, any of the following can be coded:

```
IF PRIMARY-OTHER...
  (which tests for values 1, 2, 3, 5, 6)
IF PRIMARY-FOUR...
  (which tests for value 4)
IF JUNIOR-HI...
  (which tests for values 7, 8, 9)
IF SENIOR-HI...
  (which tests for values 10, 11, 12)
```

Depending on the evaluation of the condition-name condition, alternative paths of processing are taken by the object program.

Relation Condition

A relation condition causes a comparison between two operands, either of which can be an identifier, a literal, or an arithmetic expression.

Format

```
                     { GREATER THAN }
                     { LESS THAN    }
operand-1 IS [ NOT ] { EQUAL TO     } operand-2
                     { >            }
                     { <            }
                     { =            }
```

Operand-1 is the subject of the relation condition; operand-2 is the object of the relation condition. Operand-1 and operand-2 can each be an identifier, a literal, or an arithmetic expression. The relation condition must contain at least one reference to an identifier. Except when two numeric operands are compared, operand-1 and operand-2 must have the same USAGE.

The relational operator specifies the type of comparison to be made. Table 21 on page 357 shows relational operators and their meanings. Each relational operator must be preceded and followed by a space.

Table 21. Relational Operators and Their Meanings

Relational Operator	Meaning
IS [NOT] GREATER THAN IS [NOT] >	Greater than or not greater than
IS [NOT] LESS THAN IS [NOT] <	Less than or not less than
IS [NOT] EQUAL TO IS [NOT] =	Equal to or not equal to

Rules for numeric and nonnumeric comparisons are given in the following paragraphs. If either of the operands is a group item, nonnumeric comparison rules apply. Figure 77 summarizes the permissible comparisons.

	SECOND OPERAND													
First Operand	GR	AL	AN	ANE	NE	FC NNL	ZR NL	ZD	BI	PD	AE	BO	IN	IDI
Group (GR)	NN	NN	NN	NN	NN	NN	NN	NN						
Alphabetic (AL)	NN	NN	NN	NN	NN	NN	NN	NN						
Alphanumeric (AN)	NN	NN	NN	NN	NN	NN	NN	NN						
Alphanumeric edited (ANE)	NN	NN	NN	NN	NN	NN	NN	NN						
Numeric edited (NE)	NN	NN	NN	NN	NN	NN	NN	NN						
Figurative constants, except ZERO (FC) and nonnumeric literal (NNL)	NN	NN	NN	NN	NN			NN						
Figurative constant ZERO (ZR) and numeric litreal (NL)	NN	NN	NN	NN	NN			NU	NU	NU	NU		IO*	
Zoned decimal (ZD)	NN	NN	NN	NN	NN	NN	NU	NU	NU	NU	NU		IO*	
Binary (BI)							NU	NU	NU	NU	NU		IO*	
Packed decimal (PD)							NU	NU	NU	NU	NU		IO*	
Arithmetic expression (AE)							NU	NU	NU	NU	NU			
Boolean data item (BO) or Boolean literal												BO		
Index-name (IN)							IO*	IO*	IO*	IO*			IO	IV
Index data item (IDI)													IV	IV

BO = Comparison as described for Boolean operands.
NN = Comparison as described for nonnumeric operands.
NU = Comparison as described for numeric operands.
IO = Comparison as described for two index-names (by occurrence number).
IV = Comparison as described for index data items (by value).
Blank = Comparison is not allowed.

* Valid only if the numeric item is an integer.

Figure 77. Permissible Comparisons of Operands

Note: The Boolean Data Item or Boolean Literal (First Operand row), and BO (Second Operand column) represent an IBM Extension to ANS COBOL X3.23-1974.

Comparison of Numeric Operands: For numeric class operands, algebraic values are compared. The length (number of digits) of the operands is not significant. Zero is considered a unique value, regardless of the sign. Unsigned numeric operands are considered positive; regardless of their USAGE, comparison of numeric operands is permitted.

IBM Extension

Comparison of Boolean Operands: Boolean operands can be used only in the [NOT] EQUAL TO relation condition. Boolean operands cannot be compared to non-Boolean operands. Boolean data items and literals must be one position in length. Two Boolean operands are equal if they both have a value of Boolean 1 or Boolean 0. The Boolean operands are unequal if one has a value of Boolean 1 and the other has a value of Boolean 0.

End of IBM Extension

Comparison of Nonnumeric Operands: A comparison of two nonnumeric operands or of one numeric and one nonnumeric operand is made with respect to the program collating sequence in use.

When a nonnumeric and a numeric operand are compared, the following rules apply:

- If the nonnumeric operand is a literal or an elementary data item, the numeric operand is treated as though it were moved to an alphanumeric elementary data item of the same size, and the contents of this alphanumeric data item were then compared with the nonnumeric operand.
- If the nonnumeric operand is a group item, the numeric operand is treated as though it were moved to a group item of the same size, and the contents of this group item were then compared with the nonnumeric operand. For further discussion of the rules for alphanumeric and group move operations, see the "MOVE Statement" on page 436.

Numeric and nonnumeric operands can be compared only when their USAGE, explicitly or implicitly, is the same. In such comparisons, the numeric operand must be described as an integer literal or data item; noninteger literals and data items must not be compared with nonnumeric operands.

The size of each operand is the total number of characters in that operand; the size affects the result of the comparison. There are two kinds of operands to consider: operands of equal size and operands of unequal size.

Operands of equal size: Characters in corresponding positions of the two operands are compared, beginning with the leftmost character and continuing through the rightmost character.

If all pairs of characters through the last pair test as equal, the operands are considered equal. If a pair of unequal characters is encountered, the characters are tested to determine their relative positions in the collating sequence. The operand containing the character higher in the sequence is considered the greater operand.

Operands of unequal size: If the operands are of unequal size, the comparison is made as though the shorter operand were extended to the right with enough spaces to make the operands equal in size.

Note: Valid comparisons for index-names and index data items are discussed under "Table Handling" in Chapter 11, "Using the Additional COBOL Functions."

Sign Condition

The sign condition determines whether or not the algebraic value of a numeric operand is less than, greater than, or equal to zero.

Format

```
                    { POSITIVE }
operand IS [ NOT ] { NEGATIVE }
                    { ZERO     }
```

The operand being tested must be defined as a numeric identifier or as an arithmetic expression that contains at least one reference to an identifier.

The operand is POSITIVE if its value is greater than zero, NEGATIVE if its value is less than zero, and ZERO it its value is equal to zero. An unsigned operand is POSITIVE or ZERO.

When NOT is specified, one algebraic test is processed for the truth value of the sign condition. For example, NOT ZERO is regarded as true when the operand tested is positive or negative in value.

Switch-Status Condition

The switch-status condition determines the on or off status of an UPSI switch.

Format

```
condition-name
```

The condition-name must be defined to be associated with the ON or OFF value of a switch in the SPECIAL-NAMES paragraph.

The switch-status condition tests the value associated with the condition-name. The result of the test is true if the UPSI switch is set to the position corresponding to condition-name.

Complex Conditions

A complex condition is a condition in which one or more logical operators act upon one or more conditions. Complex conditions include:

- Negated simple conditions
- Combined conditions
- Negated combined conditions.

Each logical operator must be preceded and followed by a space. The logical operators and their meanings are shown in Table 22 on page 360.

The truth value of a complex condition depends on the truth values of the simple conditions and negated simple conditions that make up the complex condition. The logical operators tell the compiler how to combine these individual truth values.

Table 22. Logical Operators and Their Meanings

Logical Operator	Meaning
AND	Logical conjunction–the truth value is true when both conditions are true.
OR	Logical inclusive OR–the truth value is true when either or both conditions are true.
NOT	Logical negation–reversal of truth value (the truth value is true if the condition is false).

Negated Simple Conditions

A simple condition is negated through the use of the logical operator NOT.

Format

```
NOT simple-condition
```

The simple-condition to be negated can be a class condition, a condition-name condition, a relation condition, a sign condition, or a switch-status condition. The simple-condition cannot be negated if the condition itself contains a NOT.

The negated simple-condition gives the opposite truth value as the simple condition. That is, if the truth value of the simple-condition is true, then the truth value of that same negated simple-condition is false.

Placing a negated simple-condition within parentheses does not change its truth value. For example, the following two statements are equivalent:

```
NOT A IS EQUAL TO B.

NOT (A IS EQUAL TO B).
```

Combined Conditions

Two or more conditions can be logically connected to form a combined condition.

Format

```
condition { { AND } condition } . . .
          { { OR  }           }
```

The condition to be combined can be a simple-condition, a negated simple-condition, a combined condition, a negated combined condition (that is, the NOT logical operator followed by a combined condition enclosed in parentheses). Combinations of the preceding conditions are specified according to the rules given in Table 23 on page 361.

Parentheses are never needed when either AND or OR (but not both) are used exclusively in one combined condition. However, parentheses might be needed to find a

final truth value when a combination of AND, OR, and NOT is used. There must a one-to-one correspondence between left and right parentheses with each left parenthesis to the left of its corresponding right parenthesis.

Table 23 summarizes the way in which conditions and logical operators can be combined and parenthesized. Figure 78 illustrates the relationships between logical operators and conditions C1 and C2 where C1 and C2 are any conditions as defined above.

Table 23. Valid Combinations of Conditions, Logical Operators, and Parentheses in a Conditional Expression

	Permissible Position in Conditional Elements			
Condition Element	**Leftmost**	**When Not Leftmost, Can Be Immediately Preceded By:**	**When Not Rightmost, Can Be Immediately Followed By:**	**Rightmost**
Simple-Condition	Yes	OR NOT AND (	OR AND)	Yes
OR AND	No	Simple-Condition)	Simple-Condition NOT (	No
NOT	Yes	OR AND (	Simple-Condition (	No
(	Yes	OR NOT AND (	Simple-Condition NOT (	No
)	No	Simple-Condition)	OR AND)	Yes

Values for C1	Values for C2	C1 AND C2	C1 OR C2	NOT (C1 AND C2)	NOT C1 AND C2	NOT (C1 OR C2)	NOT C1 OR C2
True	True	True	True	False	False	False	True
False	True	False	True	True	True	False	True
True	False	False	True	True	False	False	False
False	False	False	False	True	True	True	True

Figure 78. How Logical Operators Affect the Evaluation of Conditions

Evaluating Conditional Expressions: If parentheses are used, logical evaluation of combined conditions proceeds in the following order:

1. Conditions within parentheses are evaluated first.

2. Within nested parentheses, evaluation proceeds from the least inclusive condition to the most inclusive condition.

If parentheses are not used (or are not at the same level of inclusiveness), the combined condition is evaluated in the following order:

1. Arithmetic expressions.
2. Simple-conditions in the following order:
   a. Relation
   b. Class
   c. Condition-name
   d. Switch-status
   e. Sign.
3. Negated simple-conditions in the same order as item 2.
4. Combined conditions, in the following order:
   a. AND
   b. OR.
5. Negated combined conditions in the following order:
   a. AND
   b. OR.
6. Consecutive operands at the same evaluation-order level are evaluated from left to right. However, the truth value of a combined condition can sometimes be determined without evaluating the truth value of all the component conditions.

The component conditions of a combined condition are evaluated from left to right. If the truth value of one condition is not affected by the evaluation of further elements of the combined condition, these elements are not evaluated. However, the truth value of the condition will always be the same (as if the condition had been evaluated in full), as described earlier in this paragraph.

For example:

```
NOT A IS GREATER THAN B OR A + B IS EQUAL
  TO C AND D IS POSITIVE
```

is evaluated as if parenthesized as follows:

```
(NOT (A IS GREATER THAN B)) OR (((A+B) IS EQUAL
  TO C) AND (D IS POSITIVE))
```

The order of evaluation in this example is as follows:

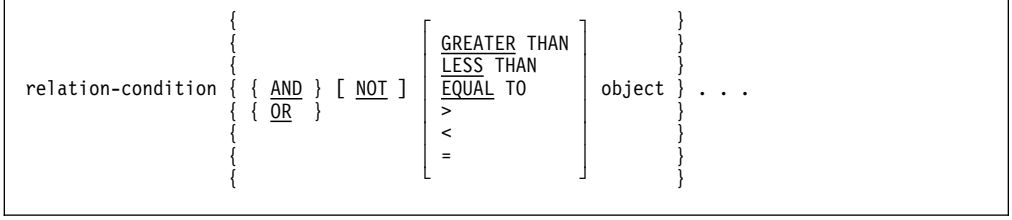
1. (NOT (A IS GREATER THAN B)) is evaluated. If true, the rest of the condition is not evaluated, as the expression is true.
2. (A+B) is evaluated, giving some intermediate result, x.
3. (x IS EQUAL TO C) is evaluated. If false, the rest of the condition is not evaluated, as the expression is false.
4. (D IS POSITIVE) is evaluated, giving the final truth value of the expression.

Abbreviated Combined Relation Conditions

When relation-conditions are written consecutively and no parentheses are used within the consecutive sequence, any relation-condition after the first can be abbreviated by either:

- Omission of the subject
- Omission of the subject and relation operator.

Format



In any consecutive sequence of relation-conditions, both forms of abbreviation can be specified. The abbreviated condition is evaluated as if:

- The last stated subject is the missing subject.
- The last stated relational operator is the missing relational operator.
- The resulting combined condition must comply with the rules for element sequence in combined conditions, as shown in Table 23 on page 361.
- The word NOT is considered part of the relational operator in the forms NOT GREATER THAN, NOT >, NOT LESS THAN, NOT <, NOT EQUAL TO, and NOT =.
- NOT in any other position is considered a logical operator, and thus results in a negated related-condition.

Table 24 shows examples of abbreviated combined relation-conditions and their nonabbreviated equivalents.

Table 24. Abbreviated Combined Relation-Condition Equivalent

Abbreviated Combined Relation-Condition	Nonabbreviated Equivalent
A = B AND NOT LESS THAN C OR D	((A = B) AND (A NOT LESS THAN C)) OR (A NOT LESS THAN D)
A NOT GREATER THAN B OR C	(A NOT GREATER THAN B) OR (A NOT GREATER THAN C)
NOT A = B OR C	(NOT (A = B) OR (A = C))
NOT (A = B OR LESS THAN C)	NOT ((A = B) OR (A LESS THAN C))
NOT (A NOT = B AND C AND NOT D)	NOT (((A NOT = B) AND (A NOT = C)) AND (NOT (A NOT = D)))

Declarative

The Declaratives section provides a method of calling procedures that are processed when an exceptional condition occurs that cannot normally be tested by the COBOL programmer. Declarative procedures are provided for the processing of exceptional input/output conditions and debugging procedures.

Format

```
[ DECLARATIVES.

{ section-name SECTION [ segment-number ] . USE statement.

[ paragraph-name. [ sentence ] . . . ] . . . } . . .

  END DECLARATIVES. ]
```

Declarative procedures are written at the beginning of the Procedure Division in a series of Declarative sections. Each such section is preceded by a USE statement that identifies under what conditions the section is used. The series of procedures that follow specify what actions are to be taken when the exceptional condition occurs. Each Declarative section ends with the occurrence of another section-name followed by a USE statement, or with the keywords END DECLARATIVES.

The entire group of Declarative procedures is preceded by the keyword DECLARATIVES, written on the next line after the Procedure Division header; the group is followed by the keywords END DECLARATIVES. The keywords DECLARATIVES and END DECLARATIVES must each begin in Area A and be followed by a period. No other text can appear on the same line.

In the Declaratives portion of the Procedure Division, each section header (with an optional segment number) must be followed by a period and a space, and must be followed by a USE statement followed by a period and a space. No other text can appear on the same line. There are two forms of the USE statement:

- USE AFTER EXCEPTION/ERROR
- USE FOR DEBUGGING.

The USE statement itself is never processed; instead, the USE statement defines the conditions that will cause processing of the immediately following procedural paragraphs, which specify the actions to be taken. After the procedure is run, control is returned to the routine that activated it.

Within a Declarative procedure, except for the USE statement itself, there must be no reference to any nondeclarative procedure.

Within a Declarative procedure, no statement can be processed that would cause the processing of a USE procedure that has been previously called and has not yet returned control to the calling routine.

An exit from a Declarative procedure is effected by processing the last statement in the procedure.

In this chapter, only the USE AFTER EXCEPTION/ERROR Declaratives procedure is described. The USE FOR DEBUGGING Declaratives procedure is described under “DEBUGGING FEATURES” on page 517.

EXCEPTION/ERROR Declarative

The EXCEPTION/ERROR Declarative specifies procedures for input/output exception or error handling that are to be run in addition to the standard system procedures.

Format

```
                                                { file-name-1 [ , file-name-2 ] . . . }
                                                { INPUT                               }
USE AFTER STANDARD { EXCEPTION } PROCEDURE ON { OUTPUT                              } .
                   { ERROR     }             { I-O                                 }
                                                { EXTEND                              }
```

The words EXCEPTION and ERROR are synonymous and can be used interchangeably.

File-Name Phrase

This phrase is valid for sequential, indexed, and relative files. When this phrase is specified, the procedure is run only for the file(s) named. No file-name can refer to a sort-merge file. For any given file, only one EXCEPTION/ERROR procedure can be specified. For example, if an input file is specifically named in one EXCEPTION/ERROR procedure, there must not also be an EXCEPTION/ERROR procedure for all INPUT files.

IBM Extension

The file-name phrase is also valid for TRANSACTION files.

End of IBM Extension

INPUT Phrase

This phrase is valid for sequential, indexed, and relative files. When this phrase is specified, the procedure is applicable to all files opened in INPUT mode.

OUTPUT Phrase

This phrase is valid for sequential, indexed, and relative files. When this phrase is specified, the procedure is applicable to all files opened in OUTPUT mode.

I-O Phrase

This phrase is valid for sequential, indexed, and relative files. When this phrase is specified, the procedure is applicable to all files opened in I-O mode.

IBM Extension

The I-O phrase is also valid for TRANSACTION files.

End of IBM Extension

EXTEND Phrase

This phrase is valid for sequential files only. When this phrase is specified, the procedure is applicable to all files opened in EXTEND mode.

General Considerations

The EXCEPTION/ERROR Declaratives procedure is run when one of the following conditions exists:

- After completing the standard system input/output error routine
- Upon recognition of an INVALID KEY or AT END condition when an INVALID KEY or AT END option has not been specified in the input/output statement
- When Status Key 1 is not equal to 0 following an I-O operation.

The EXCEPTION/ERROR Declarative procedures are processed when an input/output error occurs during processing of a READ, WRITE, REWRITE, START, DELETE, ACQUIRE, DROP, OPEN, or CLOSE statement. For example, these procedures are activated when an input/output statement fails on a file that is in the open status.

After processing the EXCEPTION/ERROR Declarative procedure, control is returned to the statement immediately following the input/output statement that caused the error.

Within a Declarative procedure, there must be no reference to any nondeclarative procedure. In the nondeclarative portion of the program, there must be no reference to procedure-names that appear in an EXCEPTION/ERROR Declarative procedure, except that PERFORM statements can refer to an EXCEPTION/ERROR Declarative procedure or to procedures associated with it.

Within an EXCEPTION/ERROR Declarative procedure, no statement can be processed that causes processing of a USE statement that has been previously called and has not yet returned control to the calling routine.

Programming Notes

EXCEPTION/ERROR Declarative procedures can be used to check the status key values whenever an input/output error occurs. Additional information about the file causing the error can be obtained by using data from the mnemonic-names OPEN-FEEDBACK and I-O-FEEDBACK.

Care should be used in specifying EXCEPTION/ERROR Declarative procedures for any file. Prior to successful completion of an initial OPEN for any file, the current Declarative has not yet been established by the object program. Therefore, if any other I-O statement is processed for a file that has never been opened, no Declarative can receive control. However, if this file has been previously opened, the last previously established Declarative procedure receives control.

For example, an OPEN OUTPUT statement establishes a Declarative procedure for this file, and the file is then closed without error. During later processing, if a logic error occurs, control will go to the Declarative procedure established when the file was opened OUTPUT.

If there is no applicable USE procedure in the program when an I-O error occurs, processing can continue. Unless the program is terminated, or some other action taken, other errors may occur, causing undesirable results.

Conditional Statements

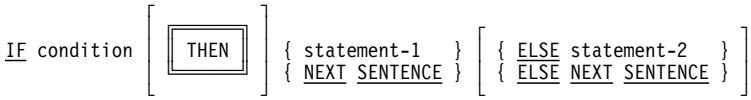
A conditional statement specifies that a truth value of a condition is to be determined, and that the subsequent action of the object program depends on this truth value. Table 16 on page 350 gives a list of the conditional statements.

Only the IF statement is discussed in this section; the other conditional statements are discussed elsewhere in this manual.

IF Statement

The IF statement causes a condition to be evaluated, and provides for alternative actions in the program, depending on that value.

Format



IBM Extension

THEN is used as a separator.

End of IBM Extension

Statement-1 or statement-2 can be any one of the following:

- An imperative statement
- A conditional statement
- An imperative statement followed by a conditional statement.

The scope of an IF statement can be terminated by any of the following:

- A separator period
- If nested, by an ELSE phrase associated with an IF statement at a higher level of nesting.

If the condition tested is true, one of the following actions takes place:

- Statement-1, if specified, is processed. If statement-1 contains a procedure branching statement, control is transferred according to the rules for that statement. If statement-1 does not contain a procedure-branching statement, the ELSE phrase, if specified, is ignored, and control passes to the next executable sentence.
- NEXT SENTENCE, if specified, is processed; that is, the ELSE phrase, if specified, is ignored, and control passes to the next executable sentence.

If the condition tested is false, one of the following:

- ELSE statement-2, if specified, is processed. If statement-2 contains a procedure-branching statement, control is transferred according to the rules for

that statement. If statement-2 does not contain a procedure-branching statement, control is passed to the next executable sentence.

- ELSE NEXT SENTENCE, if specified, is processed. Therefore, statement-1, if specified, is ignored; control passes to the next executable sentence.
- If ELSE phrase is omitted, control passes to the next executable sentence.
- The ELSE NEXT SENTENCE phrase can be omitted if it immediately precedes the period that ends the conditional sentence.

Note: When the ELSE phrase is omitted, all statements following the condition and preceding the period for the sentence are considered to be part of statement-1.

Nested IF Statements

The presence of one or more IF statements within an initial IF statement constitutes a nested IF statement.

Statement-1 and statement-2 in IF statements can consist of one or more imperative statements and/or a conditional statement. If an IF statement appears as statement-1 or as part of statement-1, it is said to be nested. Nesting statements is much like specifying subordinate arithmetic expressions enclosed in parentheses and combined in large arithmetic expressions.

IF statements contained within IF statements must be considered as paired IF and ELSE combinations, proceeding from left to right. Thus, any ELSE encountered must be considered to apply to the immediately preceding IF that has not already paired with an ELSE.

Figure 79 on page 369 shows the possible true/false combinations for the following nested IF statement:

```
IF condition-1
  statement-1-1
  IF condition-2
     IF condition-3
       statement-3-1
     ELSE
       statement-3-2
  ELSE
     statement-2-2
     IF condition-4
       IF condition-5
         statement-5-1
       ELSE
         statement-5-2.
```

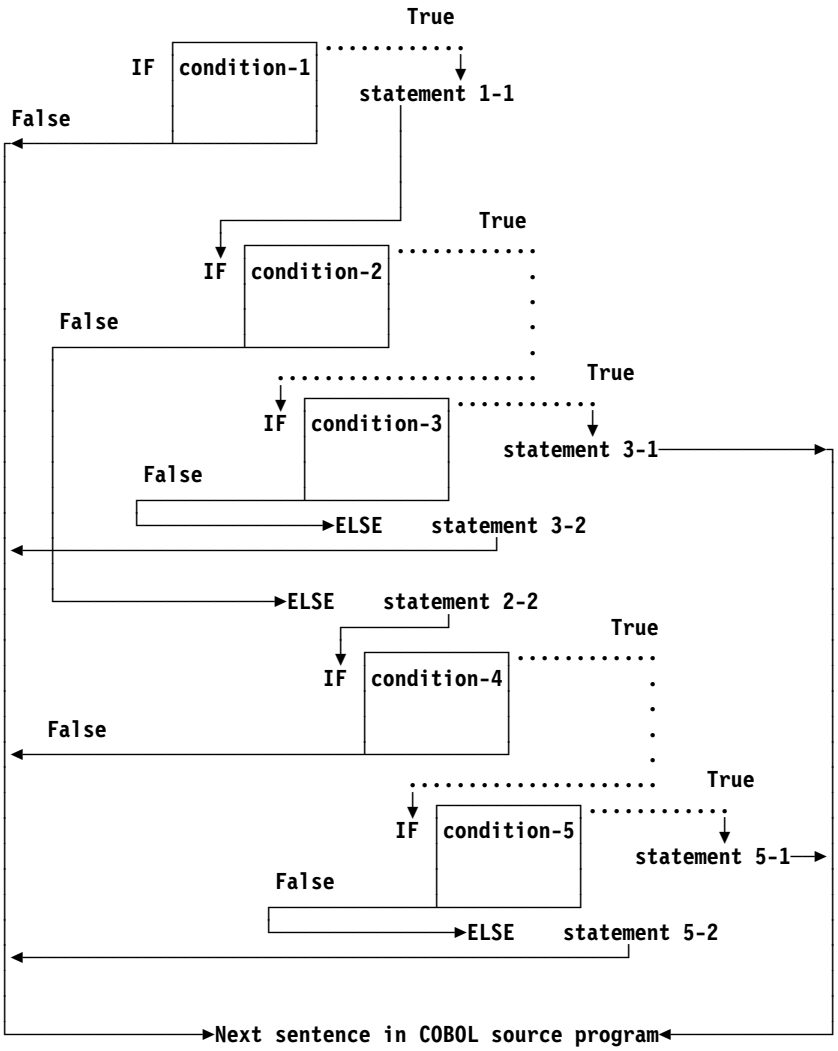


Figure 79. Nested IF Statement–True/False Combinations

Programming Notes: Because their logic is often difficult to follow, nested IF statements should, wherever possible, be avoided in a COBOL program. Often a series of simple IF statements can be used in place of the nested IF statement.

For example, the following series of simple IF statements give results equivalent to those achieved using the preceding nested IF statement example:

```
IF condition-1 NEXT SENTENCE
  ELSE GO TO PARA-2.

statement-1-1.

IF condition-2 NEXT SENTENCE
  ELSE GO TO PARA-1.

IF condition-3 statement-3-1 GO TO PARA-2
  ELSE statement-3-2 GO TO PARA-2.
```

```
PARA-1.

  statement-2-2.

  IF condition-4 NEXT SENTENCE
    ELSE GO TO PARA-2.

  IF condition-5 statement-5-1
    ELSE statement-5-2.

PARA-2.

  next-executable-statement.
```

Notice that Figure 79 on page 369 also illustrates the logic flow for the preceding series of simple IF statements.

Input/Output Statements

COBOL input/output statements transfer data to and from files. In COBOL, the unit of data made available to the program is a record, and the COBOL user need be concerned only with such records. Provision is automatically made for such operations as the movement of data into buffers and/or internal storage, validity checking, error correction (when feasible), and unblocking and blocking of records.

The description of the file in the Environment Division and the Data Division governs which input/output statements are allowed in the Procedure Division.

All TRANSACTION file formats of the input/output statements are discussed in Chapter 5, “Interactive Processing Considerations and Example Programs”

For information about COBOL file processing in relation to AS/400 file processing, see Chapter 7, “System/38-Compatible COBOL Programming Considerations.” See Appendix E, “File Structure Support Summary and Status Key Values” for a file structure support summary.

Common Input/Output Phrases

There are several phrases and concepts common to input/output statements. These are: status key, INVALID KEY condition, INTO/FROM identifier phrase, and current record pointer. The description of these phrases precedes the descriptions of the individual statements.

Status Key

If the FILE STATUS clause is specified in the file-control entry, a value is placed in the specified status key (the two-character data item named in the FILE STATUS clause) during processing of any request on that file; the value indicates the status of that request. The value is placed in the status key before processing of any EXCEPTION/ERROR Declarative or INVALID KEY/AT END phrase associated with the request.

The first character of the status key is known as status key 1; the second character is known as status key 2. Combinations of possible values and their meanings are shown in Appendix E, “File Structure Support Summary and Status Key Values.”

INVALID KEY Condition

The INVALID KEY condition can occur during processing of a START, READ, WRITE, REWRITE, or DELETE statement. When the INVALID KEY condition is recognized, the actions are taken in the following order:

1. If the FILE STATUS clause is specified in the file-control entry, a value is placed into the status key to indicate an INVALID KEY condition (see Appendix E, “File Structure Support Summary and Status Key Values.”)
2. If the INVALID KEY phrase is specified in the statement causing the condition, control is transferred to the INVALID KEY imperative-statement. Any EXCEPTION/ERROR Declarative procedure specified for this file is not processed.
3. If the INVALID KEY phrase is not specified, but an EXCEPTION/ERROR Declarative procedure is specified for the file, the EXCEPTION/ERROR procedure is run.

When an INVALID KEY condition occurs, the input/output statement that caused the condition is unsuccessful. If the INVALID KEY phrase is not specified for a file, an EXCEPTION/ERROR procedure must be specified.

INTO/FROM Identifier Phrase

This phrase is valid for READ, REWRITE, and WRITE statements. The identifier specified must be the name of an entry in the Working-Storage Section, the Linkage Section, or of a record description for another previously opened file. Record-name/file-name and identifier must not refer to the same storage area. In both phrases, an implicit move is processed according to MOVE statement rules without the CORRESPONDING phrase.

The following illustrate the use of the INTO/FROM identifier phrase in an input/output statement:

READ file-name RECORD INTO identifier.

WRITE record-name FROM identifier.

Current Record Pointer

The current record pointer identifies a particular record accessed by a sequential input request. The record identified depends on the statement being processed. The OPEN, READ, RETURN, ROLLBACK, and START statements position the current record pointer as follows:

- The OPEN statement positions the current record pointer to the first record in the file.

IBM Extension

The current record pointer can be positioned to any record in the file, at the time the OPEN is issued, by using the POSITION parameter of the Override with Data Base File (OVRDBF) command.

End of IBM Extension

- For a sequential access READ statement, or a dynamic access READ NEXT statement, the following considerations apply:
  - If an OPEN or START statement positioned the current record pointer, the record identified by the current record pointer is made available. If this record does not exist, the next existing record is made available.
  - If a previous READ statement positioned the current record pointer, the current record pointer is updated to point to the next existing record in the file; that record is then made available.

IBM Extension

- For a dynamic access READ FIRST statement, the current record pointer is positioned to point to the first record in the file; that record is then made available.
- For a dynamic access READ LAST statement, the current record pointer is positioned to point to the last record in the file; that record is then made available.
- For a dynamic access READ PRIOR statement, the current record pointer is positioned to point to the previous existing record in the file; that record is then made available.

End of IBM Extension

- For the RETURN statement, the following considerations apply:
  - The first RETURN statement positions the current record pointer to the first record in the file, and that record is then made available.
  - If a previous RETURN statement positioned the current record pointer, the current record pointer is updated to point to the next existing record in the file, and the record is then made available.
- For the ROLLBACK statement, the following considerations apply to any file under commitment control:
  - The ROLLBACK statement sets the current record pointer to the pointer's position at the previous commitment boundary. This is important to remember if you are doing sequential processing.
  - The current record pointer is set to the pointer's position at the OPEN if no COMMIT statement has been issued since the file was opened.
  - The current record pointer is undefined for any file under commitment control that is not open.
- The START statement positions the current record pointer to the first record in the file that satisfies the implicit or explicit comparison specified in the START statement.

The setting of the current record pointer is affected only by the OPEN, START, RETURN, READ, and ROLLBACK statements. The concept of the current record pointer has no meaning for files with an access mode of random, for TRANSACTION files, or for output files.

IBM Extension

DB-FORMAT-NAME Special Register

After the processing of an input/output statement, for a FORMATFILE or DATABASE file, the DB-FORMAT-NAME special register is modified according to the following rules:

- After completion of a successful READ, WRITE, REWRITE, START, or DELETE operation, the record format name used in the I-O operation is implicitly moved to the special register.
- After an unsuccessful input/output operation, DB-FORMAT-NAME contains the record format name used in the last successful input/output operation.
- DB-FORMAT-NAME is implicitly defined as PICTURE X(10).

End of IBM Extension

ACCEPT Statement

The function of the ACCEPT statement is to obtain low volume data. The processing of an ACCEPT statement causes the transfer of data into the specified identifier. There is no editing or error checking of the incoming data. The formats of the ACCEPT statement are as follows:

Format 1

```
ACCEPT identifier [ FROM mnemonic-name ]
```

Format 2

```
                        { DATE }
ACCEPT identifier FROM  { DAY  }
                        { TIME }
```

Format 3

```
ACCEPT identifier FROM mnemonic-name
  [ FOR file-name ]
```

Format 4

```
ACCEPT identifier-1 FROM mnemonic-name

****************************
*  [                       ] *
*  [ FOR { identifier-2 } ] *
*  [     { literal      } ] *
*  [                       ] *
****************************
```

Format 5–TRANSACTION Attributes

```
ACCEPT identifier-1  FROM mnemonic-name

[                                          ]
[ FOR { identifier-2 } [ FOR file-name ] ] .
[     { literal      }                     ]
[                                          ]
```

Format 1 Considerations

This format is used to transfer data from an input/output device to the identifier. Identifier can be any fixed length group item, or an elementary alphabetic, alphanumeric, or zoned decimal item.

When the FROM phrase is omitted, the ACCEPT statement obtains input from the job input stream for batch jobs, and from the work station for interactive jobs.

The job input stream is CL request data. If there is no data in the input stream, an exception occurs. See Chapter 4, “Running and Debugging Your Program” for further information on the placement of input data for a batch job.

When the FROM phrase is specified, mnemonic-name must be associated with an input/output device that is specified in the SPECIAL-NAMES paragraph. The input/output device can be the work station (REQUESTOR) or the system operator’s message queue (CONSOLE or SYSTEM-CONSOLE). If mnemonic-name is REQUESTOR and the job is a batch job, the job input stream is used.

When the input is from the job input stream, the following rules apply:

- An input record size of 80 characters is assumed.
- If the identifier is up to 80 characters in length, the input data must appear as the first character within the input record. Any characters beyond the length of identifier are truncated.
- If the identifier is longer than 80 characters, succeeding input records are read until the storage area of the identifier is filled. If the length of the identifier is not an exact multiple of 80 characters, the last input record is truncated.

When the device is the work station, the input record size is 62. When the device is the system operator's message queue, the input record size is 58. The following steps occur:

1. A system-generated inquiry message containing the program-name, the text "AWAITING REPLY FOR POSITION(S)", and the beginning and ending positions is automatically sent to the system operator's message queue or the work station operator. Previous DISPLAYs can also appear on the ACCEPT screen.
2. Processing is suspended.
3. The reply is moved into the identifier, and processing is resumed after a reply is made by the operator to the inquiry in step 1. The reply value is made available to the program as it was entered, in uppercase or lowercase.
4. If the identifier is longer than the input record size, then succeeding input records are read (steps 1-3) until the identifier is filled.

If the incoming reply is longer than the identifier, the character positions beyond the length of identifier are truncated.

The source of input data is dependent upon the type of program initiation as follows:

Method of Program Initiation	Mnemonic-Name Associated with SYSTEM-CONSOLE	Mnemonic-Name Associated with REQUESTOR	Data Source When FROM Phrase Omitted
BATCH	System operator's message queue	Job input stream	Job input stream
INTERACTIVE	System operator's message queue	Work station	Work station

Format 2 Considerations

This format is used to transfer the system date or system time to the identifier, using the rules for the MOVE statement without the CORRESPONDING phrase. Identifier can be a group item, or an elementary alphanumeric, alphanumeric edited, numeric, or numeric edited item.

IBM Extension

A numeric item can also be defined as COMPUTATIONAL-3 or COMPUTATIONAL-4.

End of IBM Extension

DATE, DAY, and TIME implicitly have USAGE DISPLAY.

DATE has the implicit PICTURE 9(6). The sequence of data elements from left to right is: two digits for year of century, two digits for month of year, two digits for day of month. Thus July 4, 1976 is expressed as 760704.

DAY has the implicit PICTURE 9(5). The sequence of data elements from left to right is: two digits for year of century, three digits for day of year. Thus, July 4, 1976 is expressed as 76186.

TIME has the implicit PICTURE 9(8). The sequence of data elements from left to right is: two digits for hour of day, two digits for minute of hour, two digits for second of minute, two digits for hundredths of second. Thus 12.25 seconds after 2:41 p.m. is expressed as 14411225.

Format 3 Considerations

This format is used to transfer feedback information from an active file to the identifier. The identifier can be any fixed-length group item or an elementary alphabetic, alphanumeric, or zoned decimal item. The file must be defined in an FD entry, and must be open prior to the processing of the ACCEPT statement. If the file is not open, the contents of identifier remain unchanged.

The FROM phrase specifies a mnemonic-name that must be associated with a function-name of OPEN-FEEDBACK or I-O-FEEDBACK in the SPECIAL-NAMES paragraph.

When the FOR phrase is specified, the feedback information is from the file specified in the phrase. When the FOR phrase is not specified, the feedback information is from the last file opened or used in an input or output operation.

See Appendix E, “File Structure Support Summary and Status Key Values” for a discussion of the I-O-FEEDBACK and OPEN-FEEDBACK areas. See the *System/38 CPF Programmer's Guide* for a layout and description of the data areas contained in the feedback areas.

Format 4 Considerations

This format is used to transfer data to identifier-1 from the system-defined local data area created for a job.

This format is only applicable when the mnemonic-name in the SPECIAL-NAMES paragraph is associated with the function-name LOCAL-DATA.

The move into identifier-1 takes place according to the rules for the MOVE statement for a group move without the CORRESPONDING phrase.

When the FOR phrase is specified, it is syntax checked during compilation but treated as comments during processing. The value of literal or identifier-2 indicates the program device name of the device that is associated with the local data area. There is only one local data area for each job, and all devices in a job access the same local data area. Literal, if specified, must be nonnumeric and ten characters or less in length. Identifier-2, if specified, must refer to an alphanumeric data item, ten characters or less in length.

See “Local Data Area” on page 266 for more information.

Format 5 Considerations

See “ACCEPT Statement” on page 126 for a discussion of Format 5.

Programming Notes

The Format 1 ACCEPT statement is useful for exceptional situations in a program when operator intervention (to supply a given message, code, or exception indicator) is required. The operator must, of course, be supplied with the appropriate messages with which to reply.

The Format 2 ACCEPT statement allows the programmer access to the current date (in two formats) and time of day, as carried by the system. This can be useful in identifying when a particular run of a program was processed. It can also be used to supply the date in headings, footings, and so on.

IBM Extension

ACQUIRE Statement

The ACQUIRE statement acquires a program device for a TRANSACTION file. See Chapter 5, “Interactive Processing Considerations and Example Programs” on page 89 for a discussion of this statement and the format shown below.

Format

```
ACQUIRE { identifier } FOR file-name
        { literal    }
```

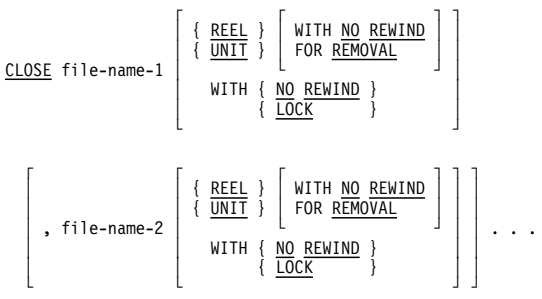
See “ACQUIRE Statement” on page 127 for a discussion of this format.

End of IBM Extension

CLOSE Statement

The CLOSE statement terminates the processing of volumes and files. REWIND, LOCK, and REMOVAL phrases are specified, as applicable. The formats of the CLOSE statement are as follows:

Format 1



Format 2

```
CLOSE file-name-1 [ WITH LOCK ]
  [ file-name-2 [ WITH LOCK ] ] . . .
```

See “CLOSE Statement” on page 128 for a discussion of Format 2.

Each file-name designates a file upon which the CLOSE statement is to operate. These files:

- Need not have the same organization or access mode
- Must not be sort or merge files.

A CLOSE statement without the REEL/UNIT phrase can be successfully processed for a file. In this case, an OPEN statement for the file must be processed before any other input/output statement can refer explicitly or implicitly to the file. This is true for all input/output statements except a SORT/MERGE statement with the USING or GIVING phrases.

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the CLOSE statement is processed.

If the file is open and the processing of a CLOSE statement is unsuccessful, the EXCEPTION/ERROR procedure for the file, if specified, is run.

If a CLOSE statement for an open file is not processed before a STOP RUN for this run unit, the file is implicitly closed.

If a CANCEL statement is processed for a program with an open file:

- The status of that file is unpredictable
- The file can be logically damaged
- The file can keep the allocated device longer than necessary.

If the SELECT OPTIONAL clause is specified in the file-control entry for this file and the file is not present at run time, standard end-of-file processing is not performed.

The following tables illustrate organization, access, device, and volume considerations for the CLOSE statement. The letter codes used in the tables are defined in the section following the tables.

Note: Card devices are not supported by System/38-Compatible COBOL, even though the devices are accepted by the syntax checker.

Sequential Organization

ACCESS	S E Q U E N T I A L								
DEVICE	READER	PUNCH	PRINT	PUNCHPRINT	PRINTER	DISKETTE	DISK	DATABASE	FORMATFILE
CLOSE	K, J	K, J	K, J	K, J	K, J	K , J	K, J	K, J	K, J
CLOSE WITH LOCK	K, J, E	K, J, E	K, J, E	K, J, E	K, J, E	K, J, E	K, J, E	K, J, E	K, J, E
REEL/UNIT	-	-	-	-	-	-	-	-	-
REMOVAL	-	-	-	-	-	-	-	-	-
NO REWIND	-	-	-	-	-	-	-	-	-

Sequential Organization

ACCESS	SEQUENTIAL	
DEVICE	TAPEFILE	
VOLUME	SINGLE	MULTI
CLOSE	K, J, G	K, J, G, A
CLOSE WITH LOCK	K, J, G, E	K, J, G, A, E
CLOSE NO REWIND	K, J, B	K, J, B, A
CLOSE REEL/UNIT	C	K, F, G
CLOSE REEL/UNIT FOR REMOVAL	C	K, F, D, G
CLOSE REEL/UNIT WITH NO REWIND	C	K, F, B

Indexed Organization

ACCESS	ANY	
DEVICE	DISK	DATABASE
CLOSE	K, J	K, J
CLOSE WITH LOCK	K, J, E	K, J, E
REEL/UNIT	-	-
REMOVAL	-	-
NO REWIND	-	-

Relative Organization

ACCESS	ANY	
DEVICE	DISK	DATABASE
CLOSE	K, J	K, J
CLOSE WITH LOCK	K, J, E	K, J, E
REEL/UNIT	-	-
REMOVAL	-	-
NO REWIND	-	-

Letter Code	Meaning
–	An invalid combination.
A	No effect on any previous volumes. Any additional volumes are not processed.
B	The current volume is left in its present position. The reel is not rewound.

IBM Extension

The system always rewinds and unloads the tape when REEL/UNIT is specified on the CLOSE statement.

End of IBM Extension

C Optional, but only syntax-checked (performs no function at run time).

D The current volume is rewound and unloaded. The system is notified that the volume is logically removed from this run unit. However, the volume can be accessed again, after processing of a CLOSE statement without the REEL/UNIT phrase and an OPEN statement for this file.

E COBOL ensures that this file cannot be reopened during this processing of the program.

F Close volume procedures. The labels are handled as follows:

	LABEL RECORDS	
MODE OF FILE	STANDARD	OMITTED
Input	F01	F02
Output	F03	F04
I-O	F01	F02

F01 The current volume is positioned to read the labels. If this is the last volume for the file, the next processed READ statement receives the AT END condition. If this is not the last volume, the following actions are taken:

1. The current volume is unloaded
2. The beginning standard labels on the next volume are read
3. The next processed READ statement gets the first record on the newly mounted volume.

F02 The current volume is unloaded. If all of the reels as specified on the REELS parameter of the Create Tape File (CRTTAPF) or Override with Tape File (OVRTAPF) CL command have been processed, the next processed READ statement receives the AT END condition. If there are more reels, the next volume is mounted, and the next processed READ statement gets the first record on the newly mounted volume.

F03 The standard end-of-volume labels are written. The next volume is mounted. The standard beginning labels are written on the new volume. The next processed WRITE statement places the next logical record on the newly mounted volume.

F04 The system end-of-volume procedures for nonlabeled tapes are run. The next volume is mounted. The system beginning of volume procedures for nonlabeled tapes are run. The next processed WRITE statement places the next logical record on the newly mounted volume.

G The current volume is positioned at its beginning.

J The record area associated with the file-name is no longer available after successful processing of this statement. Unsuccessful processing of this statement leaves availability of the record data area undefined.

Labels are processed as follows:

	LABEL RECORDS	
MODE OF FILE	STANDARD	OMITTED
Input	J01	J02
Output	J03	J04
I-O	J01	J02

J01 If the file is positioned at its end, the label records are read and verified, and the file is closed. If the file is not at its end, the file is closed.

J02 The file is closed.

J03 The standard label records are written, and the file is closed.

J04 The file is closed without any label processing.

K May be processed only for an open file.

IBM Extension

COMMIT Statement

The COMMIT statement provides a way of synchronizing changes to data base records while preventing other jobs from modifying those records until the COMMIT is processed. The format of the COMMIT statement is:

Format

When the COMMIT statement is processed, all changes made to files under commitment control since the previous commitment boundary are made permanent. A commitment boundary is established by the successful processing of a ROLLBACK or COMMIT statement. If no COMMIT or ROLLBACK has been issued in the current job, a commitment boundary is established by the first OPEN of any file under commitment

control in the job. Changes are made to all files under commitment control in the job, not just to files under commitment control in the COBOL program that issues the COMMIT statement.

When a COMMIT is processed, all record locks held by the job since the last commitment boundary for files under commitment control are released and the records become available to other jobs.

The COMMIT statement only affects files under commitment control. If a COMMIT is processed and there are no files opened under commitment control, the COMMIT statement has no effect and no commitment boundary is established.

The COMMIT statement does *not*:

- Modify the I-O-FEEDBACK area for any file
- Change the current record pointer for any file
- Set a file status value for any file.

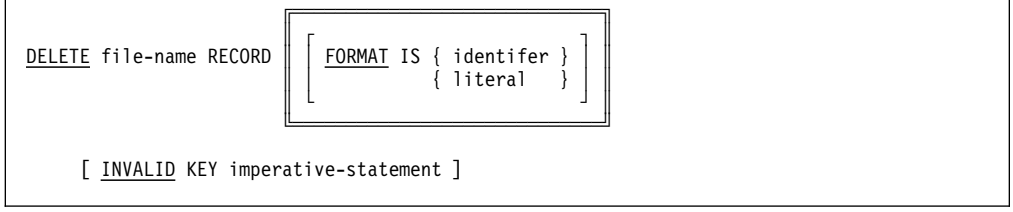
For more information on commitment control, see “Commitment Control Considerations” on page 247.

End of IBM Extension

DELETE Statement

The DELETE statement logically removes a record from an indexed or relative file. The format of the DELETE statement is as follows:

Format



After successful processing of a DELETE statement, the record is logically removed from the file. It is no longer accessible. Processing of the DELETE statement does not affect the contents of the record area associated with file-name.

If the FILE STATUS clause is specified in the FILE-CONTROL entry, the associated status key is updated when the DELETE statement is processed.

The current record pointer is not affected by the processing of the DELETE statement.

The following tables illustrate organization, access, and device considerations for the DELETE statement. The letter codes used in the tables are defined in the section following the tables.

Sequential Organization

	Device	Any
	Access	SEQUENTIAL
DELETE Verb		Not Allowed

Relative Organization

	Device	DISK			DATABASE		
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
DELETE Verb		A, P, Z	B, P, Z	B, P, Z	A, P, Z	B, P, Z	B, P, Z
INVALID KEY		-	O, U	O, U	-	O, U	O, U
FORMAT		-	-	-	-	-	-

Indexed Organization

	Device	DISK			DATABASE		
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
DELETE Verb		A, P, Z	D, P, Z	D, P, Z	A, P, Z	D, P, Z	D, P, Z
INVALID KEY		-	O, U	O, U	-	O, U	O, U
FORMAT		-	-	-	-	S, F	S, F

Letter Code	Meaning
–	An invalid combination.
A	The last input/output statement must have been a successfully processed READ statement. When the DELETE statement is processed, the system logically removes the record retrieved by that READ statement.
B	The system logically removes the record identified by the contents of the RELATIVE KEY data item. If the file does not contain such a record, an INVALID KEY condition exists. The space is then available for a new record with the same RELATIVE KEY value.
D	The system logically removes the record identified by the contents of the RECORD KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

IBM Extension

When EXTERNALLY-DESCRIBED-KEY is specified for the file, the key field in the record area[15] for the format specified by the FORMAT phrase is used to find the record to be deleted. If the FORMAT phrase is not specified, the first format defined in the program for the file is used to find the record to be deleted.

[15] The key field in the record area is the location in the buffer selected in accordance with a record format or specification in order to build a search argument.

If the DUPLICATES phrase was specified for this file, the last input/output statement processed for this file before processing of the DELETE statement must have been a successfully processed READ statement. The record read by that statement is the record that is deleted.

In this case, the FORMAT phrase is not used to find the record to be deleted. The READ statement is required to ensure that the proper record is deleted when there are duplicates. If a successful read operation did not occur before the delete operation:

- The file status key, if defined, is set to 94.
- The USE AFTER STANDARD EXCEPTION/ERROR procedure, if specified, is run.
- The delete operation is not processed.

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format must be operated on.

The literal or identifier must be a character-string of ten characters or less. If an identifier is specified, it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is called, if applicable for the file.

End of IBM Extension

O Optional.

P Allowed when the file is opened for I-O.

S Required when processing a file that has multiple record formats and has unique keys.

U The INVALID KEY phrase must be specified for files in which an applicable USE procedure is not specified.

IBM Extension

Z The action of this statement can be inhibited at program run time by the inhibit write (INHWRT) parameter of the Override with Data Base File (OVRDBF) CL command. When this parameter is specified, non-zero file status codes are not set for data dependent errors. Duplicate key and data conversion errors are examples of data dependent errors.

See the *CL Reference* for more information on this command.

End of IBM Extension

DISPLAY Statement

The DISPLAY statement transfers low-volume data to an output device.

Format 1

```
DISPLAY { identifier-1 } [ , identifier-2 ]
        { literal-1    } [ , literal-2    ]

        ... [ UPON mnemonic-name ]
```

Format 2

```
DISPLAY { identifier-1 } [ { , identifier-2 } ] . . .
        { literal-1    } [ { , literal-2    } ]

  UPON   mnemonic-name

**************************
* [ FOR { identifier-3} ] *
* [     { literal-3   } ] *
**************************
```

Format 1 Considerations

The DISPLAY statement transfers the contents of each operand to the output device in the left-to-right order in which the operands are listed. When a DISPLAY statement is processed, the data contained in the sending field is transferred to the output device. The size of the sending field is the total character count of all operands listed. If the total character count is less than the maximum logical record size, the remaining rightmost characters are padded with spaces. If the total character count exceeds the maximum, as many records are written as are needed to display all operands. Any operand being displayed when the end of a record is reached is continued in the next record.

Numeric identifiers not described as USAGE IS DISPLAY are converted automatically to zoned decimal.

IBM Extension

COMPUTATIONAL-4 items are also converted to zoned decimal. Signed noninteger numeric literals are allowed.

End of IBM Extension

Signed values in numeric fields cause the last character to show both the sign and number. For example, if SIGN WITH SEPARATE CHARACTER is not specified and two numeric items have the values -34 and 34, they are displayed as 3M and 34,

respectively. If SIGN WITH SEPARATE CHARACTER is specified, a + or a - sign is displayed as either leading or trailing, depending on how the number was specified.

Note: Group items containing packed or binary data (COMP, COMP-3, or COMP-4) should not be displayed on a display station. Such data can contain display station control characters which can cause undesirable and unpredictable results.

A literal can be any figurative constant except the ALL literal. When a figurative constant is specified as one of the operands, only a single occurrence of the figurative constant is displayed.

When the UPON phrase is omitted, the DISPLAY statement sends output to the REQUESTOR. When the UPON phrase is specified, mnemonic-name must be associated in the SPECIAL-NAMES paragraph with either the work station (REQUESTOR) or the system operator's message queue (CONSOLE or SYSTEM-CONSOLE).

The record length depends on the device as follows:

Output	Maximum Logical Record Size
Job log	120 characters
Work station	58 characters
System operator's message queue	58 characters

When a program in a batch job processes a DISPLAY statement without the UPON phrase, or with an UPON phrase associated with the REQUESTOR, the output is sent to the job log in an informational message of severity 80. You can change the severity of this message using the Change Message Description (CHGMSGD) CL command.

For more information, see the *CL Reference*.

For an interactive job that uses display device files, DISPLAY statements are not normally used. If you do use them, the following considerations apply.

When an interactive job processes a DISPLAY statement, the logical record appears on the screen in the Program Messages display.

The following screen shows an example of a Program Messages display.

```
                          Display Program Messages
Job E41.QPGMR.004541 started 08/23/88 13:06:39
THIS IS AN EXAMPLE DISPLAY STATEMENT
AND THIS IS ANOTHER
COBOL STOP literal in program SAMPDISP (C G)









Type reply, press Enter.
  Reply . . .

F3=Exit   F12=Previous
```

This display contains messages from the current program processing, as well as messages relating to other activities in the session.

The display device file on the screen when a DISPLAY statement is processed determines whether program processing is suspended as a result of the DISPLAY statement processing.

- If the parameter RSTDSP(*NO) is specified when the display device file is changed or created (CHGDSPF or CRTDSPF command), DISPLAY statement processing suspends program processing, and the Program Messages display appears on the screen. You must press the Enter key to resume program processing. The previous display returns to the screen immediately.
- If the parameter RSTDSP(*YES) is specified when the display device file is changed or created (CHGDSPF or CRTDSPF command), DISPLAY statement processing does not suspend program processing. The Program Messages display appears on the screen, and remains on the screen until one of the following happens:
  - The program processes a nonsubfile READ or WRITE statement for that file. The Program Messages Display then disappears, and the display device file is returned to the screen.
  - The program terminates.

  Note: If you want to suspend program processing, code an ACCEPT statement after the DISPLAY statement. Program processing is suspended until you press the Enter key.

To view output records after the program terminates, press the F10 key from the Command Entry display.

For additional information on interactive processing, see Chapter 5, “Interactive Processing Considerations and Example Programs” For additional information on the RSTDSP parameter, see the CHGDSPF and CRTDSPF commands in the *CL Reference*.

When a program started by a work station operator sends a DISPLAY to the system operator’s station (separate from the work station), program processing is not suspended.

The location of the output data is dependent upon the type of program initiation as follows:

Method of Initiation	Mnemonic-Name Associated with SYSTEM-CONSOLE	Mnemonic-Name Associated with REQUESTOR	UPON Phrase Omitted
BATCH	System operator’s message queue	Job log	Job log
INTERACTIVE	System operator’s message queue	Work station	Work station

IBM Extension

Format 2 Considerations

This format is used to transfer data to the system-defined local data area created for a job.

This format is only applicable when the mnemonic-name in the SPECIAL-NAMES paragraph is associated with the function name LOCAL-DATA.

The DISPLAY statement’s literal operands, or the contents of the DISPLAY statement’s identifier operands, are written to the system-defined local data area of the job containing the program that issues the DISPLAY. The data is written to the local data area according to the rules of the MOVE statement for a group move, without the CORRESPONDING phrase.

The FOR phrase, when specified, is syntax checked during compilation but is treated as comments during processing. The value of literal-3 or identifier-3 indicates the program device name of the device that is writing data to the local data area. There is only one local data area for each job, and all devices in a job access the same local data area. Literal-3, if specified, must be nonnumeric and ten characters or less in length, and identifier-3, if specified, must refer to an alphanumeric data item ten characters or less in length.

For more information, see “Local Data Area” on page 266.

End of IBM Extension

IBM Extension

DROP Statement

The DROP statement releases a program device that was acquired by a TRANSACTION file.

Format

```
DROP { identifier }  FROM file-name
     { literal    }
```

See “DROP Statement” on page 129 for a discussion of this format.

End of IBM Extension

OPEN Statement

The OPEN statement initiates file processing. It also checks and/or writes labels. The formats of the OPEN statement are as follows:

Format 1–Sequential Files

```
     {                                                    }
     { INPUT file-name-1 [ REVERSED          ]            }
     {                   [ WITH NO REWIND    ]            }
     {                                                    }
     {     [ , file-name-2 [ REVERSED          ] ] . . .  }
     {     [               [ WITH NO REWIND    ] ]        }
OPEN {                                                    } . . .
     {                                                    }
     { OUTPUT file-name-3 [ WITH NO REWIND ]              }
     {                                                    }
     {     [ , file-name-4  [ WITH NO REWIND ] ] . . .    }
     {                                                    }
     { I-O file-name-5 [ , file-name-6 ] . . .            }
     {                                                    }
     { EXTEND file-name-7 [ , file-name-8 ] . . .         }
```

Format 2–Indexed and Relative Files

```
     {                                                 }
     { { INPUT  }                                      }
OPEN { { OUTPUT } file-name-1 [ , file-name-2 ] . . . } . . .
     { { I-O    }                                      }
     {                                                 }
```

Format 3–TRANSACTION Files

```
OPEN I-O file-name-1 [ file-name-2 ] . . .
```

See “OPEN Statement” on page 129 for a discussion of Format 3.

Each file-name designates a file upon which the OPEN statement is to operate. The files specified need not have the same organization or access. Each file-name must be defined in an FD entry in the Data Division, and must not name a sort or merge file. The FD entry must be compatible with the information supplied to the system when the file was defined.

At least one of the phrases (INPUT, OUTPUT, I-O, or EXTEND) must be specified. These phrases can appear in any order. More than one file name can be specified in each phrase.

A file can be opened for INPUT, OUTPUT, I-O, or EXTEND in the same program. After the first OPEN statement is processed for a file, each subsequent OPEN statement processing must be preceded by a successful CLOSE file statement processing without the LOCK phrase.

The following tables illustrate organization, access, and device considerations for the OPEN statement. The letter codes used in the tables are defined in the section following the tables.

Note: Card devices are not supported by the System/38-Compatible COBOL, even though the devices are accepted by the syntax checker.

Sequential Organization

	ACCESS	SEQUENTIAL									
	DEVICE	READER	PUNCH	PRINT	PUNCHPRINT	PRINTER	TAPEFILE	DISKETTE	DISK	DATABASE	FORMATFILE
OPEN Verb		S	S	S	S	S	L, S	S	S	S, C	S
INPUT		R, A, F	-	-	-	-	O, A, F, L1, N	O, A, F, N	O, A, K,F,N	O, A, K, F, N	O, A, K, F
OUTPUT		-	R, J	R, J	R, J	R, J	O, J, L2, N	O, J, N	O, G, N	O, G, N	O, G
I-O		-	-	-	-	-	-	-	O, M, K	O, M, K	O, M, K
NO REWIND		-	-	-	-	-	O, D	-	-	-	-
REVERSED		-	-	-	-	-	O, B	-	-	-	-
EXTEND		-	-	-	-	-	O, E, L3	-	O, E	O, E	-

Relative Organization

	Device	DISK			DATABASE		
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
OPEN Verb		H, S	H, S	H, S	H, S, C	H, S, C	H, S, C
INPUT		O, A, K, N	O, A	O, A, K	O, A, K, N	O, A	O, A, K
OUTPUT		O, G, N	O, G	O, G	O, G, N	O, G	O, G
I-O		O, M, K	O, M	O, M, K	O, M, K	O, M	O, M, K
NO REWIND		-	-	-	-	-	-
REVERSED		-	-	-	-	-	-
EXTEND		-	-	-	-	-	-

Indexed Organization

	Device	DISK			DATABASE		
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
OPEN Verb		S	S	S	S, C	S, C	S, C
INPUT		O, A, K, N	O, A	O, A, K	O, A, K, N	O, A	O, A, K
OUTPUT		O, G, N	O, G	O, G	O, G, N	O, G	O, G
I-O		O, M, K	O, M	O, M, K	O, M, K	O, M	O, M, K
NO REWIND		-	-	-	-	-	-
REVERSED		-	-	-	-	-	-
EXTEND		-	-	-	-	-	-

Letter Code **Meaning**

– An invalid combination.

A The file is opened for input operations. The current record pointer is set to the first record in the file. If no records exist in the file, the current record pointer is set so that processing of the first sequential READ statement results in an AT END condition.

B OPEN statement processing positions the file at its end. Subsequent READ statements make the data records available in reverse order, starting with the last record. REVERSED can only be specified for input files.

IBM Extension

C The file may be be placed under commitment control.

See “Commitment Control Considerations” on page 247 for more information.

End of IBM Extension

D The OPEN statement does not reposition the file. The tape must be positioned at the beginning of the desired file before processing of the OPEN statement.

IBM Extension

The system keeps track of the current position on the tape and automatically positions the tape to the proper place. When processing a multifile tape volume, all CLOSE statements should specify the LEAVE phrase. When the next file on the volume is opened, the system determines which direction the tape should be moved to most efficiently get to the desired file.

End of IBM Extension

E The EXTEND phrase permits opening the file for output operations. OPEN EXTEND statement processing prepares the file for the addition of records. These additional records immediately follow the last record in the file.

Subsequent WRITE statements add records as if the file had been opened for OUTPUT. The EXTEND phrase can be specified when a file is being created.

F If SELECT OPTIONAL is specified in the file-control entry, OPEN statement processing causes the program to check for the presence or absence of this file at run time. If the file is absent, the first READ statement for this file causes the AT END condition to occur.

IBM Extension

G Only a physical file is cleared when opened for OUTPUT. When the file is successfully opened, it contains no records. If an attempt is made to open a logical file for OUTPUT, the file is opened but no records are deleted. The file is treated as though the EXTEND phrase had been specified. To clear a logical file, all the members on which the logical file is based should be cleared.

End of IBM Extension

H Not allowed for logical file members:

- That are based on more than one physical file.
- That contain select/omit logic.

I Allowed when the file is opened for INPUT.

J The file is opened to allow only output operations. When the file is successfully opened, it contains no records.

IBM Extension

K The first record to be made available to the program can be specified at run time by using the POSITION parameter on the OVRDBF CL command. See the *CL Reference* for more information on this command.

End of IBM Extension

L When label records are specified but not present, or when label records are present but not specified, processing of the OPEN statement can have unpredictable results.

L1 The beginning labels are checked.

L2 The labels are checked, then new labels are written.

L3 The following results occur:

- Beginning file labels are processed only if this is a single-volume file.
- Beginning volume labels of the last existing volume are checked.
- The file is positioned to the existing ending file labels. The labels are checked and then deleted.
- Processing continues as if the file were opened as an output file.

M The file is opened for both input and output operations. The current record pointer is set to the first record in the file. If no records exist in the file, the current record pointer is set so that processing of the first sequential READ statement results in an AT END condition.

N The compiler generates code to block output records or unblock input records if the conditions listed in "Unblocking Input Records and Blocking Output Records" on page 210 are satisfied.

O Optional.

R Required.

S The successful processing of an OPEN statement determines the availability of the file and results in that file being open. Before successful processing of the OPEN statement for a file, no statement, except a SORT or MERGE statement with the USING or GIVING phrase, that refers explicitly or implicitly to that file can be processed. The successful processing of the OPEN statement makes the associated record area available to the program. It does not obtain or release a data record.

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the OPEN statement is processed.

If an OPEN statement is issued for a file that is already open, the EXCEPTION/ERROR procedure for this file, if specified, is processed. See Appendix E, "File Structure Support Summary and Status Key Values" for the file status codes.

READ Statement

At run time, the READ statement makes a record available before processing of any statement following the READ statement.

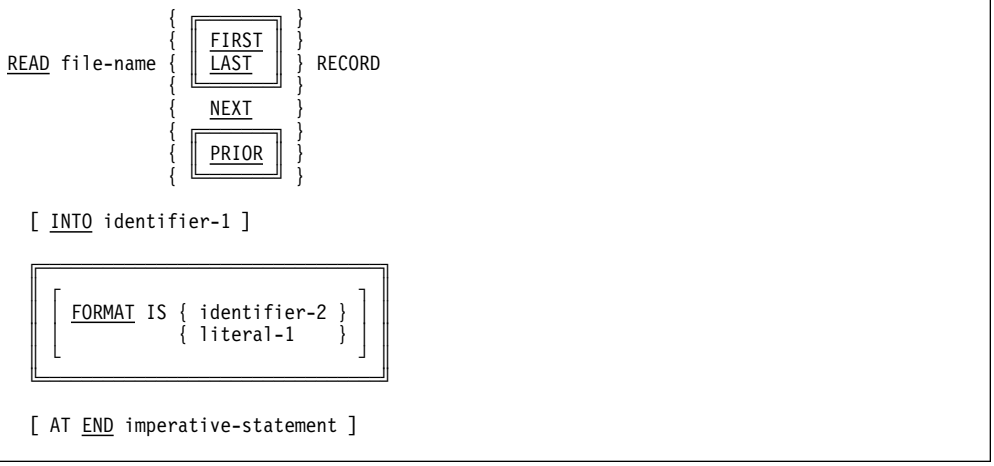
For sequential access, the READ statement makes available the next logical record from a file. For random access, the READ statement makes available a specified record from a file.

The formats for the READ statement are as follows:

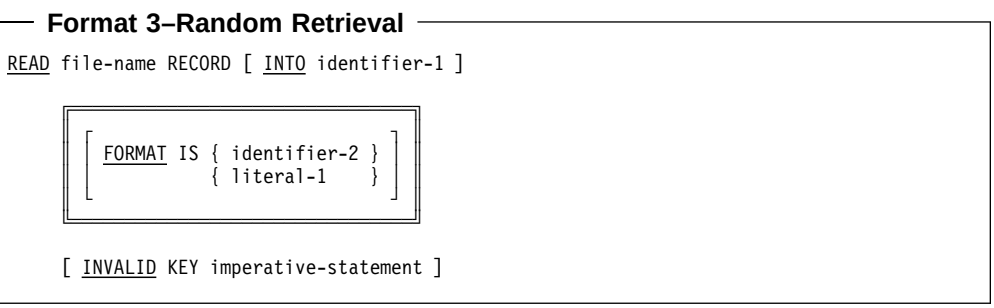
Format 1–Sequential Retrieval Using SEQUENTIAL Access

```
READ file-name RECORD

  [ INTO identifier-1 ]

  [ FORMAT IS { identifier-2 } ]
              { literal-1    }

  [ AT END imperative-statement ]
```

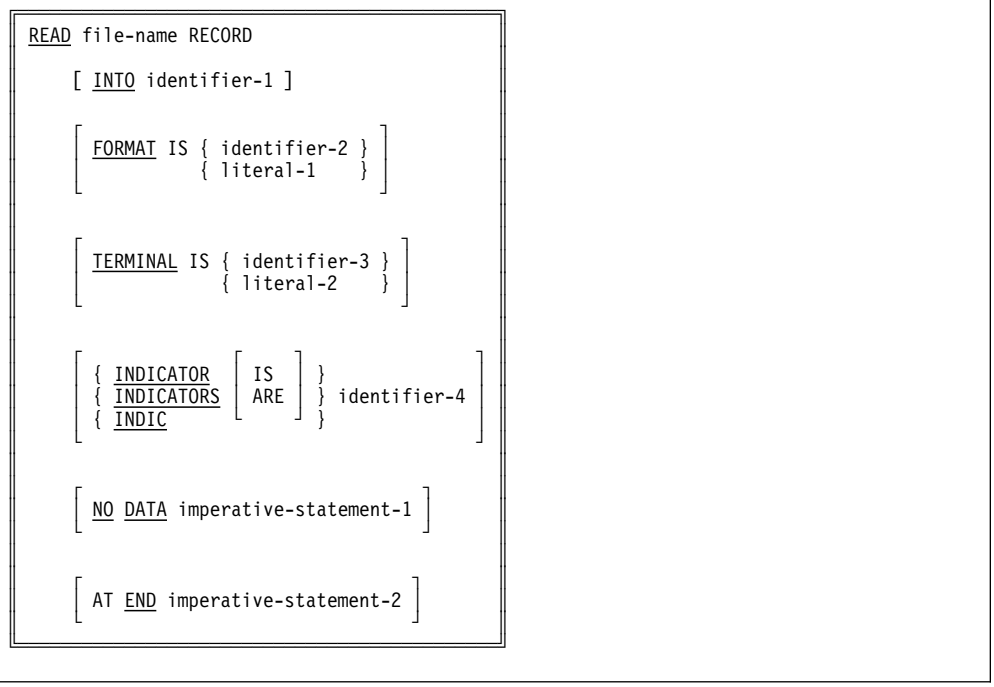
Format 2–Sequential Retrieval Using DYNAMIC Access



Format 3–Random Retrieval



Format 4–TRANSACTION File (Nonsubfile)



Format 5–TRANSACTION File (Subfile)

```
READ SUBFILE file-name

    [ NEXT MODIFIED ] RECORD

    [ INTO identifier-1 ]

    [ FORMAT IS { identifier-2 } ]
                { literal-1    }

    [ TERMINAL IS { identifier-3 } ]
                  { literal-2    }

    [ { INDICATOR  [ IS  ] }                  ]
    [ { INDICATORS [ ARE ] } identifier-4     ]
    [ { INDIC              }                  ]

    [ INVALID KEY imperative-statement-1 ]

    [ AT END imperative-statement-2 ]
```

See “READ Statement” on page 131 for a discussion of Format 4 and 5.

File-name must be defined in a Data Division FD entry, and must not name a sort or merge file. If more than one record-description entry is associated with file-name, these records automatically share the same storage area. That is, they are implicitly redefined.

After a READ statement is processed, only those data items within the range of the current record are replaced. Data items stored beyond this range are undefined. Figure 80 on page 396 illustrates this concept.

The FD entry is:

```
FD   INPUT-FILE LABEL RECORDS OMITTED.
01   RECORD-1 PICTURE X(30).
01   RECORD-2 PICTURE X(20).
```

Contents of the input area before the READ statement is processed:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

Contents of the record being read in (RECORD-2):

```
01234567890123456789
```

Contents of the input area after the READ statement is processed:

```
0123456789012345678?????????
```

(Characters in the input area represented by question marks are undefined.)

Figure 80. READ Statement with Multiple Record Descriptions

The following tables illustrate organization, access, and device considerations for the READ statement. The letter codes used in the tables are defined in the section following the tables.

Note: Card devices are not supported by System/38-Compatible COBOL, even though the devices are accepted by the syntax checker.

Sequential Organization

ACCESS	S E Q U E N T I A L									
DEVICE	READER	PUNCH	PRINT	PUNCHPRINT	PRINTER	TAPEFILE	DISKETTE	DISK	DATABASE	FORMATFILE
READ Verb	A, I, G1, N	-	-	-	-	A, I, G1, N, V	A, I, G1, N, V	A, I, P, G1, N	A, I, P, G1, N	A, I, G1, N
NEXT	-	-	-	-	-	-	-	-	-	-
LAST	-	-	-	-	-	-	-	-	-	-
FIRST	-	-	-	-	-	-	-	-	-	-
PRIOR	-	-	-	-	-	-	-	-	-	-
INTO	O, B	-	-	-	-	O, B	O, B	O, B	O, B	O, B
AT END	O, E, D1	-	-	-	-	O, E, D1	O, E, D1	O, E, D1	O, E, D1	O, E, D1
INVALID KEY		-	-	-	-	-	-	-	-	-
FORMAT		-	-	-	-	-	-	-	-	-

Relative Organization

Device	DISK			DATABASE		
Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
READ Verb	A, I, P, G2	A, I, P, G3	A, I, P	A, I, P, G2	A, I, P, G3	A, I, P
NEXT	-	-	O, Z1	-	-	O, Z1
FIRST	-	-	-	-	-	-
LAST	-	-	-	-	-	-
PRIOR	-	-	-	-	-	-
INTO	O, B	O, B	O, B	O, B	O, B	O, B
AT END	O, E, D2	-	O, E, D2	O, E, D1	-	O, E, D2
INVALID KEY	-	O, U	O, U	-	O, U	O, U
FORMAT	-	-	-	-	-	-

Indexed Organization

Device	DISK			DATABASE		
Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
READ Verb	A, I, P, G4	A, I, P, G5	A, I, P	A, I, P, G4	A, I, P, G5	A, I, P
NEXT	-	-	O, Z2	-	-	O, Z3
FIRST	-	-	-	-	-	O, Z3
LAST	-	-	-	-	-	O, Z3
PRIOR	-	-	-	-	-	O, Z3
INTO	O, B	O, B	O, B	O, B	O, B	O, B
AT END	O, E, D2	-	O, E, D2	O, E, D1	-	O, E, D2
INVALID KEY	-	O, U	O, U	-	O, U	O, U
FORMAT	-	-	-	O, F, X	O, F, W	O, F, Y

Letter Code	Meaning
–	An invalid combination.
A	If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the READ statement is processed. Following the unsuccessful processing of any READ statement, the contents of the associated record area and the position of the current record pointer are undefined.
B	The INTO identifier phrase makes a READ statement equivalent to: READ file-name RECORD MOVE record-name TO identifier After successful processing of the READ statement, the current record becomes available both in the record-name and identifier. When the INTO identifier phrase is specified, the current record is moved from the input area to the identifier area according to the rules for the MOVE statement without the CORRESPONDING phrase. Any subscripting or indexing associated with identifier is evaluated after the record has been read and immediately before it is transferred to identifier. The INTO identifier phrase cannot be specified when the file contains records of various sizes, as indicated by their record descriptions. The storage area associated with identifier and the record area associated with the file-name cannot be the same storage area.
D1	When the AT END condition is recognized, a successful CLOSE statement, followed by a successful OPEN statement, must be processed for this file before processing a READ statement.
D2	When the AT END condition is recognized, a sequential access READ statement for this file must not be processed without first processing one of the following: • A successful CLOSE statement followed by a successful OPEN statement. • A successful START statement for this file. • A successful random access READ statement for this file. • A successful READ file-name FIRST or READ file-name LAST where permitted.
E	If no next logical record exists in the file when the READ statement is processed, an AT END condition occurs, and READ statement processing is unsuccessful. The following actions are taken, in the order listed: 1. If the FILE STATUS clause is specified, the status key is updated to indicate an AT END condition. 2. If the AT END phrase is specified, control is transferred to the AT END imperative-statement. Any EXCEPTION/ERROR procedure for this file is not run. 3. If the AT END phrase is not specified, any EXCEPTION/ERROR procedure for this file is run. The AT END phrase must be specified if no explicit or implicit EXCEPTION/ERROR procedure is specified for this file.

IBM Extension

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on.

The literal or identifier must be a character-string of ten characters or less. If an identifier is specified, it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

A value of all blanks is treated as though the FORMAT phrase was not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is called, if applicable for the file.

End of IBM Extension

G1 The record that is made available by the READ statement is determined as follows:

- If the current record pointer was set by the processing of an OPEN statement, the record pointed to is made available.
- If the current record pointer was set by the processing of a previous READ statement, the pointer is updated to point to the next existing record in the file. That record is then made available.

G2 The record that is made available by the READ statement is determined as follows:

- If the current record pointer was set by the processing of a START or OPEN statement, the record pointed to is made available if it is still accessible through the path indicated by the current record pointer. If the record is no longer accessible (due, for example, to deletion of the record), the current record pointer is updated to indicate the next existing record in the file. That record is then made available.
- If the current record pointer was set by the processing of a previous READ statement, the current record pointer is updated to point to the next existing record in the file. That record is then made available.

If the RELATIVE KEY phrase is specified for this file, READ statement processing updates the RELATIVE KEY data item to indicate the relative record number of the record being made available.

G3 The record with the relative record number contained in the RELATIVE KEY data item is made available. If the file does not contain such a record, the INVALID KEY condition exists, and READ statement processing is unsuccessful.

G4 The record made available by the READ statement is determined as follows:

- If the current record pointer was set by the processing of a START or OPEN statement, the record pointed to is made available if it is still accessible through the path indicated by the current record pointer. If the record is no longer accessible (due, for example, to deletion of the

record), the current record pointer is updated to indicate the next existing record in the file. That record is then made available.

- If the current record pointer was set by the processing of a previous READ statement, the current record pointer is updated to point to the next existing record in the file. That record is then made available.

IBM Extension

For a file that allows duplicate keys (the DUPLICATES phrase is specified in the file-control entry), the records with duplicate key values are made available in the order specified when the file was created. The system options are first-in first-out (FIFO), last-in first-out (LIFO), and 'no specific sequence'.

End of IBM Extension

G5 The record in the file with a key value equal to that of the RECORD KEY data item is then made available. If the file does not contain such a record, the INVALID KEY condition exists, and READ statement processing is unsuccessful.

IBM Extension

For a file that allows duplicate keys (the DUPLICATES phrase is specified in the file-control entry), the first record with the specified key value is made available. The first record is determined by the order specified when the file was created. The system options are first-in first-out (FIFO), last-in first-out (LIFO), and 'no specific sequence'.

End of IBM Extension

I Allowed when the file is opened for INPUT.

N If SELECT OPTIONAL is specified in the file-control entry for this file and the file is not available when this program runs, processing of the first READ statement causes an AT END condition. Since the file is not available, the standard system end-of-file processing is not done when the file is closed.

O Optional.

P Allowed when the file is opened for I-O.

U The INVALID KEY phrase must be specified for files in which an appropriate USE procedure is not specified.

V If end of volume is recognized during processing of a READ statement and logical end of file has not been reached, the following actions are taken in the order listed:

1. The standard ending volume label procedure is processed.
2. A volume switch occurs.
3. The standard beginning volume label procedure is run.
4. The first data record of the next volume is made available.

The program receives no indication that the above actions occurred during the read operation.

W If specified, the key as defined for the specified format is used to get a record of that format. If a record of that format is not found, a record-not-found condition is returned. This occurs even when there are records that have the defined key, but that have a different record format.

If the format is omitted, the common key for the file is used to get the first record of any format that has that common key value. The common key for a file consists of the key fields common to all formats of a file for records residing on the data base. The common key for a file is the left-most key fields that are common across all record formats in the file. The common key is built from the data in the record description area using the first record format defined in the program for the file.

X If specified, the next record in the keyed sequence access path that has the requested format is made available. If omitted, the next record in the keyed sequence access path is made available.

Y

	FORMAT Phrase	
	Specified	Omitted
NEXT	Y1	Y2
PRIOR	Y3	Y4
FIRST	Y5	Y6
LAST	Y7	Y8
None of the Above	Y9	Y10

Y1 The next record in the keyed sequence access path having the specified format is made available.

Y2 The next record in the keyed sequence access path is made available regardless of its format.

Y3 The record in the keyed sequence access path preceding the record identified by the current record pointer having the specified format is made available.

Y4 The record in the keyed sequence access path preceding the record identified by the current record pointer is made available regardless of its format.

Y5 The first record in the keyed sequence access path having the specified format is made available.

Y6 The first record in the keyed sequence access path is made available regardless of its format.

Y7 The last record in the keyed sequence access path having the specified format is made available.

Y8 The last record in the keyed sequence access path is made available regardless of its format.

Y9 The key as defined for the specified format is used to get a record of that format. If a record of that format is not found, a record-not-found condition is returned. This occurs even when there are records that have the defined key, but that have a different record format.

Y10 The common key for the file is used to get the first record of any format that has that common key value. The common key for a file consists of the key fields common to all formats of a file for records residing on the data base. The common key for a file consists of the leftmost key fields that are common across all record formats in the file. The common key is built from the data in the record description area using the first record format defined in the program for the file.

Z1 When specified, a sequential read is done (see G2). When omitted, a random read is done (see G3).

Z2 When specified, a sequential read is done (see G4). When omitted, a random access read is done (see G5).

Z3 When specified, a sequential read is done (see G4). If NEXT, FIRST, LAST and PRIOR are all omitted, a random access read is done (see G5).

REWRITE Statement

The REWRITE statement logically replaces an existing record in a file. When the REWRITE statement is processed, the associated file must be open. The formats for the REWRITE statement are as follows:

Format 1

```
REWRITE record-name [ FROM identifier-1 ]

        [ FORMAT IS { identifier-2 } ]
                    { literal-1    }

     [ INVALID KEY imperative-statement ]
```

Format 2–TRANSACTION

```
REWRITE SUBFILE record-name [ FROM identifier-1 ]

     FORMAT IS { identifier-2 }
               { literal-1    }

     [ TERMINAL IS { identifier-3 } ]
                   { literal-2    }

     [ { INDICATOR  [ IS  ] }                    ]
     [ { INDICATORS [ ARE ] } identifier-4       ]
     [ { INDIC              }                    ]

     [ INVALID KEY imperative-statement ]
```

Record-name:

- Must be the name of a record in the File Section
- Must have the same number of character positions as the record being replaced
- Must not be subscripted or indexed
- Can be qualified.

After successful processing of a REWRITE statement, the logical record is no longer available in record-name unless the associated file is named in a SAME RECORD AREA clause. In this case, the record is also available as a record of the files named in the SAME RECORD AREA clause.

The current record pointer is not affected by processing of the REWRITE statement.

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the REWRITE statement is processed.

The following tables illustrate organization, access, and device considerations for the REWRITE statement. The letter codes used in the tables are defined in the section following the tables.

Note: Card devices are not supported by System/38-Compatible COBOL, even though the devices are accepted by the syntax checker.

Sequential Organization

ACCESS	SEQUENTIAL									
DEVICE	READER	PUNCH	PRINT	PUNCHPRINT	PRINTER	TAPEFILE	DISKETTE	DISK	DATABASE	FORMATFILE
REWRITE Verb	-	-	-	-	-	-	-	A, P	A, P	-
FROM	-	-	-	-	-	-	-	O, B	O, B	-
INVALID KEY	-	-	-	-	-	-	-	-	-	-
FORMAT	-	-	-	-	-	-	-	-	-	-

Relative Organization

Device	DISK			DATABASE		
Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
REWRITE Verb	A, P, Z	S, P, Z	S, P, Z	A, P, Z	S, P, Z	S, P, Z
FROM	O, B	O, B	O, B	O, B	O, B	O, B
INVALID KEY	-	U, H, J	U, H, J	-	U, H, J	U, H, J
FORMAT	-	-	-	-	-	-

Indexed Organization

Device	DISK			DATABASE		
Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
REWRITE Verb	A, E, P, Z	D, P, Z	D, P, Z	A, E, P, Z	D, P, Z	D, P, Z
FROM	O, B	O, B	O, B	O, B	O, B	O, B
INVALID KEY	U, G, J	U, H, J	U, H, J	U, G, J	U, H, J	U, H, J
FORMAT	-	-	-	-	M, F	M, F

Letter Code	Meaning
–	An invalid combination.
A	The last input/output statement processed for this file must have been a successfully processed `READ` statement. When the `REWRITE` statement is processed, the record retrieved by that `READ` statement is replaced.

B	The `FROM identifier` phrase makes a `REWRITE` statement equivalent to:
	`MOVE identifier TO record-name`
	`REWRITE record-name.`
	After successful processing of the `REWRITE` statement, the current record is no longer available in `record-name`, but is still available in identifier. `Record-name` and `identifier` cannot both refer to the same storage area.

D	The record to be replaced is specified by the value in the `RECORD KEY` data item. If the file does not contain such a record, an `INVALID KEY` condition exists.

IBM Extension

When EXTERNALLY-DESCRIBED-KEY is specified for the file, the key field in the record area for the format specified by the FORMAT phrase (or, if not specified, the first format defined in the program for the file), is used to find the record to be replaced.

If the DUPLICATES phrase was specified for this file, the last input/output statement processed for this file before the REWRITE statement must have been a successfully processed READ statement. The record read by that statement is the one that is replaced. In this case, the FORMAT phrase is not used in determining the record to be replaced.

The READ statement is required to ensure that the proper record is replaced when there are duplicates. If a successful read operation did not occur before the rewrite operation:

- The file status key, if defined, is set to 94.
- The EXCEPTION/ERROR procedure, if any, is run.
- The REWRITE statement is not processed.

Note: The only way to rewrite one of a sequence of records with duplicate keys is to sequentially read each of the duplicate records and rewrite the desired one.

End of IBM Extension

E The value of the RECORD KEY data item must not have been changed since the record was read.

IBM Extension

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on.

The literal or identifier must be a character-string of ten characters or less. If an identifier is specified, it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is called, if applicable for the file.

End of IBM Extension

G Processed when the value contained in the RECORD KEY of the record to be replaced does not equal the RECORD KEY data item of the last retrieved record from the file.

H Processed when the record specified by the key field in the record area is not found.

J When an INVALID KEY condition exists, the updating operation does not take place. The data in record-name is unaffected.

M Optional when processing a file that has one record format.

O Optional.

P Allowed when the file is opened for I-O.

S The record to be replaced is specified by the value in the RELATIVE KEY data item. If the file does not contain such a record, an INVALID KEY condition exists.

U The INVALID KEY phrase must be specified for files in which an applicable USE procedure is not specified.

IBM Extension

Z The action of this statement can be inhibited at program run time by the inhibit write (INHWRT) parameter of the Override with Data Base File (OVRDBF) CL command. When this parameter is specified, nonzero file status codes are not set for data dependent errors. Duplicate key and data conversion errors are examples of data dependent errors.

See the *CL Reference* for more information on this command.

End of IBM Extension

IBM Extension

ROLLBACK Statement

The ROLLBACK statement provides a way to cancel one or more changes to data base records when the changes should not remain permanent. The format of the ROLLBACK statement is:

Format

When the ROLLBACK statement is processed, any changes made to files under commitment control since the last commitment boundary are removed from the data base.[16] A commitment boundary is the previous occurrence of a ROLLBACK or COMMIT statement. If no COMMIT or ROLLBACK has been issued, the commitment boundary is the first OPEN of a file under commitment control. Removal of changes takes place for all files under commitment control in the job, and not just for files under commitment control in the COBOL program that issues the ROLLBACK.

[16] When a file is cleared while being opened for OUTPUT, processing of a ROLLBACK statement does not restore cleared records to the file.

Once the ROLLBACK is successfully processed, all record locks held by the job for files under commitment control are released and the records become available to other jobs.

The ROLLBACK has no effect on files not under commitment control. If a ROLLBACK is processed and there are no files under commitment control, the ROLLBACK is ignored.

A file under commitment control can be opened or closed without affecting the status of changes made since the last commitment boundary. A COMMIT must still be issued to make the changes permanent. A ROLLBACK, when processed, leaves files in the same open or closed state as before processing.

The ROLLBACK statement does *not*:

- modify the I-O-FEEDBACK area for any file
- set a file status value for any file.

For the ROLLBACK statement, the following considerations apply:

- The ROLLBACK statement sets the current record pointer to the pointer's position at the previous commitment boundary. This is important to remember if you are doing sequential processing.
- If no COMMIT statement has been issued since the file was opened, the ROLLBACK statement sets the current record pointer to the pointer's position at the OPEN.
- The current record pointer is undefined after a ROLLBACK if the file is closed with uncommitted changes.

At the end of every job, an implicit ROLLBACK of uncommitted records is automatically done for all files under commitment control. Any uncommitted changes to the data base are canceled.

End of IBM Extension

START Statement

The START statement provides a way of positioning within an indexed or relative file for subsequent sequential retrieval. This positioning is achieved by comparing the key values of records in the file with the value you place in the RECORD KEY portion of a file's record area (for an indexed file), or in the RELATIVE KEY data item (for a relative file) prior to processing of the START statement. The format for the START statement is as follows:

Format

```
START file-name [ KEY IS { EQUAL TO      }                                                    ]
                [        { =             }                                                    ]
                [        { GREATER THAN  } { EXTERNALLY-DESCRIBED-KEY                       } ]
                [        { >             } {                                                } ]
                [        { NOT LESS THAN } {   data-name-1 [ , data-name-2 ] . . .           } ]
                [        { NOT <         } {                                                } ]

     [ FORMAT IS { identifier-1 } ]
                 { literal-1    }

     [ INVALID KEY imperative-statement ]
```

File-name must be defined in an FD entry in the Data Division. It must not name a sort or merge file.

Data-name-1, data-name-2... can be qualified.

If the FILE STATUS clause is specified in the FILE-CONTROL paragraph, the associated status key is updated when the START statement is processed.

The following tables illustrate organization, access, and device considerations for the START statement. The letter codes used in the tables are defined in the section following the tables.

Sequential Organization

	Access	SEQUENTIAL
	Device	Any
START Verb		Not Allowed

Relative Organization

	Device	DISK			DATABASE		
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
START Verb		I, P, A	-	I, P, A	I, P, A	-	I, P, A
KEY IS		O, E	-	O, E	O, E	-	O, E
INVALID KEY		O, U, D	-	O, U, D	O, U, D	-	O, U, D
FORMAT		-	-	-	-	-	-

Indexed Organization

	Device	DISK			DATABASE		
	Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
START Verb		I, P, B	-	I, P, B	I, P, B, K	-	I,P,B,K
KEY IS		O, G	-	O, G	O, G	-	O, G
INVALID KEY		O, U, D	-	O, U, D	O, U, D	-	O, U, D
FORMAT		-	-	-	O, F, J	-	O, F, J

Letter Code	Meaning
–	An invalid combination.
A	When the KEY phrase is not specified, the current record pointer is set to the record in the file with a key (relative record number) equal to the RELATIVE KEY data item.
B	When the KEY phrase is not specified, the current record pointer is set to the record with a key equal to the value contained in the RECORD KEY data item.
D	If the comparison is not satisfied by any record in the file, an INVALID KEY condition exists. The position of the current record pointer is undefined, and the INVALID KEY imperative-statement, if specified, is processed.
E	When the KEY phrase is specified, data-name-1 must specify the RELATIVE KEY. The current record pointer is positioned to the first logical record currently existing in the file with a key (relative record number) that satisfies the comparison with the RELATIVE KEY data item.

IBM Extension

F	The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on. The literal or identifier must be a character-string of ten characters or less. If an identifier is specified, it must be the name of one of the following: • A Working-Storage Section entry • A Linkage Section entry • A record-description entry for a previously opened file. A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is called, if applicable for the file.

End of IBM Extension

G	When the KEY phrase is specified, the search argument used for the comparison is data-name-1, which can be: • The RECORD KEY itself. • An alphanumeric data item within a record description for the file with a leftmost character position that corresponds to the leftmost character position of the key field in the record area. This data item must be less than or equal to the length of the RECORD KEY for the file. This data item can be qualified. **Note:** If the RECORD KEY is defined as COMP, COMP-3, or COMP-4, the key data item must be the RECORD KEY itself. A partial key field in the record area cannot be used. The current record pointer is positioned to the first record in the file with a record key for a format that satisfies the comparison. If the operands in the comparison are of unequal length, the comparison proceeds as if the

longer field were truncated on the right to the length of the shorter field. All other numeric and nonnumeric comparison rules apply, except that the PROGRAM COLLATING SEQUENCE, if specified, has no effect.

IBM Extension

For a file that specified RECORD KEY IS EXTERNALLY-DESCRIBED-KEY, the following additional considerations apply:

- The reserved word EXTERNALLY-DESCRIBED-KEY can be specified. This indicates that the complete key field in the record area should be used in the comparison.
- A series of data names can be specified. This allows a partial key field in the record area to be used (generic START). These data names must follow the following rules:
  - All except the last of the data names specified must be a record key for a format that was copied in for the file. The record format in which they are contained does not have to be the one that can be specified by the FORMAT phrase.
  - The order of these data names (key fields) must match the order of the keys as defined in DDS; that is, they must be specified from most significant field to least significant.
  - The total number of data names cannot exceed the number of key fields defined for that record format.
  - If the last data name specified in the series is not a key field in the record area, it must have its left byte occupy the same space as the key field that is defined at that relative position. If the key field in the record area at this position is a COMP, COMP-3, or COMP-4 field, only the key field itself can be used as the data name.
- The following table shows the action between the KEY IS phrase and the FORMAT phrase:

	KEY Phrase		
FORMAT Phrase	Data-Name Series	Omitted	EXTERNALLY-DESCRIBED-KEY
Yes	G1, G2	G3, G4	G3, G2
No	G1, G5	G6, G7	G6, G5

G1 The search argument is built using the specified data items.

G2 The current record pointer is set to the first record in the file of the format specified with a record key that satisfies the comparison specified in the key phrase.

G3 The search argument is built using the key fields in the record area for the format specified in the FORMAT phrase.

G4 The current record pointer is set to the first record in the file of the specified format with a record key equal to the search argument.

G5 The current record pointer is set to the first record in the file with a common key for the file that satisfies the comparison specified in the KEY phrase. If there is no common key, the current record pointer is set to the first record in the file.

G6 The search argument is built using the key fields in the record area for the first record format for the file as defined in the program.

G7 The current record pointer is set to the first record in the file with a common key for the file that is equal to the search argument. If there is no common key, the current record pointer is set to the first record in the file.

End of IBM Extension

I Allowed when the file is opened for INPUT.

J If specified, the current record pointer is set to the first record of the specified record format that satisfies the comparison. If omitted, the current record pointer is set to the first record of any format that satisfies the comparison.

See the table in G above for a description of how this interacts with EXTERNALLY-DESCRIBED-KEY and the KEY IS phrase.

IBM Extension

K The meaning of the comparison can be affected by the type of key fields in the record area defined for the file. Key fields on this system can be defined as multiple fields, each of which can be in ascending or descending sequence. The system establishes a sequence (keyed sequence access path) for the records based on the values contained in the record key for the format and the sequencing specified in DDS. When a START statement is processed, the request is interpreted as follows:

COBOL Comparison	System Result
GREATER THAN	AFTER
NOT LESS THAN	EQUAL TO or AFTER

For example, when a statement is processed using the comparison of GREATER THAN, a search is made of these sequenced records for the first record after the search argument specified by the START statement. If the file was sequenced using descending keys, the current record pointer would point to a record with a key less than the one specified and not greater than that specified in the START statement.

End of IBM Extension

O Optional.

P Allowed when the file is opened for I-O.

U The INVALID KEY phrase must be specified for files in which an appropriate USE procedure is not specified.

WRITE Statement

The WRITE statement releases a record to the system. The formats for the WRITE statement are as follows:

Format 1–Sequential Files

```
WRITE record-name [ FROM identifier-1 ]

    [                                                    ]
    [                     { { identifer-2 } { LINE  } }  ]
    [                     { { integer     } { LINES } }  ]
    [ { BEFORE } ADVANCING {                             } ]
    [ { AFTER  }           { { mnemonic-name }           } ]
    [                      { { PAGE          }           } ]
    [                                                    ]

    [ AT { END-OF-PAGE } imperative-statement ]
    [    { EOP         }                      ]
```

Format 2–Indexed and Relative Files

```
WRITE record-name [ FROM identifier-1 ]

    [ FORMAT IS { identifier-2 } ]
    [           { literal-1    } ]

  [ INVALID KEY imperative-statement ]
```

Format 3–FORMATFILE Files

```
WRITE record-name [ FROM identifier-1 ]

    [ FORMAT IS { identifier-2 } ]
    [           { literal-1    } ]

      [ { INDICATOR  [ IS  ] }                    ]
      [ { INDICATORS [ ARE ] } identifier-3       ]
      [ { INDIC              }                    ]

      [ AT { END-OF-PAGE } imperative-statement ]
      [    { EOP         }                      ]
```

Format 4–TRANSACTION File (Nonsubfile)

```
WRITE record-name [ FROM identifer-1 ]

        FORMAT IS { identifier-2 }
                  { literal-1    }

        [ TERMINAL IS { identifier-3 } ]
                      { literal-2    }

        [ STARTING AT LINE { identifier-4 } ]
                           { literal-3    }

[ { BEFORE } ROLLING [ LINES ] { identifier-5 }                  ]
[ { AFTER  }         [ LINE  ] { literal-4    }                  ]
                                                                 ]
          [ THROUGH ] { identifier-6 } { UP   }                  ]
          [ THRU    ] { literal-5    } { DOWN }                  ]
                                                                 ]
          { identifier-7 } [ LINES ]                             ]
          { literal-6    } [ LINE  ]                             ]

        [ { INDICATOR  [ IS  ] }                                 ]
        [ { INDICATORS [ ARE ] } identifier-8                    ]
        [ { INDIC              }                                 ]
```

Format 5–TRANSACTION File (Subfile)

```
WRITE SUBFILE record-name [ FROM identifer-1 ]

    FORMAT IS { identifier-2 }
              { literal-1    }

    [ TERMINAL IS { identifier-3 } ]
                  { literal-2    }

    [ { INDICATOR  [ IS  ] }                    ]
    [ { INDICATORS [ ARE ] } identifier-4       ]
    [ { INDIC              }                    ]

    [ INVALID KEY imperative-statement ]
```

See “WRITE Statement” on page 140 for a discussion of Format 4 and 5.

Record-name:

- Must be the name of a record in the File Section of the Data Division
- Can be qualified
- Cannot be associated with a sort or merge file.

The maximum record size for a data base file is established at the time the file is defined to the system (using the Create Physical File (CRTPF) or the Create Logical File (CRTLF) CL command) and cannot be changed. If the record length defined in the program is incompatible with the record length defined to the system, the following occurs during output to the file:

- When the program record length is greater than the length defined to the system, the records are truncated to the system length. If the file is empty, the program record length is used.
- When the program record length is less than the record length defined in the system, the records are padded with blanks to make them the size specified in the system.

Processing of the WRITE statement releases a record to the file associated with record-name. After processing of a WRITE statement, the record is no longer available in record-name unless either of the following is true:

- The associated file is named in a SAME RECORD AREA clause. In this case, the record is also available as a record of the files named in the SAME RECORD AREA clause.
- The WRITE statement is unsuccessful due to a boundary violation (beyond extent).

If either of the above conditions is true, the record is still available in record-name.

The current record pointer is not affected by processing of the WRITE statement.

The number of character positions required to store the record in a file can be, but is not necessarily, the same as the number of character positions defined by the description of the record in the COBOL program. (See “PICTURE Clause” on page 332 and “USAGE Clause” on page 322.)

If the FILE STATUS clause is specified in the file-control entry, the associated status key is updated when the WRITE statement is processed.

When an attempt is made to write beyond the externally defined boundaries of the file, WRITE statement processing is unsuccessful, and an EXCEPTION/ERROR condition exists. The status key, if specified, is updated. If an explicit or implicit EXCEPTION/ERROR procedure is specified for the file, the procedure is run. If no such procedure is specified, the results are unpredictable.

The following tables illustrate organization, access, and device considerations for the WRITE statement. The letter codes used in the tables are defined in the section following the tables.

Note: Card devices are not supported by System/38-Compatible COBOL, even though the devices are accepted by the syntax checker.

Sequential Organization

ACCESS	S E Q U E N T I A L									
DEVICE	READER	PUNCH	PRINT	PUNCHPRINT	PRINTER	TAPEFILE	DISKETTE	DISK	DATABASE	FORMATFILE
WRITE Verb	-	Q, T	Q, T	Q, T	Q, T	V, Q, S, T	V, Q, T	Q, S, T, Z	Q, S, T, Z	Q, T
FROM	-	O, B	O, B	O, B	O, B	O, B	O, B	O, B	O, B	O, B
INVALID KEY	-	-	-	-	-	-	-	-	-	-
ADVANCING	-	O, D2	O, D2	O, D2	O, D1	-	-	-	-	-
AT END-OF-PAGE	-	-	-	-	O, E1	-	-	-	-	O, E2
FORMAT	-	-	-	-	-	-	-	-	-	N, F
INDICATORS	-	-	-	-	-	-	-	-	-	I

Relative Organization

Device	DISK			DATABASE		
Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
WRITE Verb	Q, K, Z	P,Q,M,Z	P,Q,M,Z	Q, K, Z	P,Q,M,Z	P,Q,M,Z
FROM	O, B	O, B	O, B	O, B	O, B	O, B
INVALID KEY	O, J1, J, U	O, J1, J2,J,U	O, J1, J2,J,U	O, J1, J, U	O, J1, J2,J, U	O, J1, J2,J,U
ADVANCING	-	-	-	-	-	-
AT END-OF-PAGE	-	-	-	-	-	-
FORMAT	-	-	-	-	-	-
INDICATORS	-	-	-	-	-	-

Indexed Organization

Device	DISK			DATABASE		
Access	SEQUENTIAL	RANDOM	DYNAMIC	SEQUENTIAL	RANDOM	DYNAMIC
WRITE Verb	G,H1,Q,Z	G, H2, P, Q, Z	G, H2, P, Q, Z	G, H1, Q, Z	G, H2, P, Q, Z	G, H2, P, Q, Z
FROM	O, B	O, B	O, B	O, B	O, B	O, B
INVALID KEY	O, J1, J4, J, U	O, J1, J3,J,U	O, J1, J3,J,U	O, J1, J4, J, U	O, J1, J3,J, U	O, J1, J3,J,U
ADVANCING	-	-	-	-	-	-
AT END-OF-PAGE	-	-	-	-	-	-
FORMAT	-	-	-	N, F	N, F	N, F
INDICATORS	-	-	-	-	-	-

Letter Code	Meaning
–	An invalid combination.
A	When the KEY phrase is not specified, the current record pointer is set to the record in the file with a key (relative record number) equal to the RELATIVE KEY data item.
B	The FROM identifier phrase makes a WRITE statement equivalent to: MOVE identifier TO record-name WRITE record-name. After successful processing of the WRITE statement, the current record is no longer available in record-name, but is still available in identifier. Record-name and identifier cannot both refer to the same storage area.
D1	When not specified, a default of AFTER ADVANCING 1 LINE is used. When specified, the following rules apply: • When BEFORE ADVANCING is specified, the line is printed before the page advances. • When AFTER ADVANCING is specified, the page advances before the line is printed. • When identifier-2 is specified, the page advances the number of lines equal to the current value in identifier-2. Identifier-2 must name an elementary integer data item. Identifier-2 can be zero. • When integer is specified, the page advances the number of lines equal to the value of integer. Integer can be zero. • When a mnemonic-name is specified, a page eject or space suppression occurs. The mnemonic-name must be equated with function-name-1 in the SPECIAL-NAMES paragraph. This phrase is not valid if a LINAGE clause is specified in the FD entry for this file. • When PAGE is specified, the record is printed on the logical page BEFORE or AFTER (depending on the phrase specified) the device is positioned to the next logical page. If PAGE has no meaning for the device used, BEFORE or AFTER ADVANCING 1 LINE is provided (depending on the phrase specified). If the FD entry contains a LINAGE clause, the device is positioned to the first printable line of the next page, as specified in that clause. If the LINAGE clause is omitted, the device is positioned to line 1 of the next page. If the LINAGE clause is specified for this file, the associated LINAGE-COUNTER special register is modified during the processing of the WRITE statement, according to the following rules: – If ADVANCING PAGE is specified, LINAGE-COUNTER is reset to 1. – If ADVANCING identifier-2 or integer is specified, LINAGE-COUNTER is incremented by the value of identifier-2 or integer. – If the ADVANCING phrase is omitted, LINAGE-COUNTER is incremented by 1. – When the device is repositioned to the first printable line of a new page, LINAGE-COUNTER is reset to 1.

D2 When not specified, stacker 1 is selected.

IBM Extension

The ADVANCING phrase using a mnemonic name can be specified to control the stacker selection. The mnemonic name must be equated with function-name-1 in the SPECIAL-NAMES paragraph and must be one of the valid stacker selection names.

End of IBM Extension

E1 The keywords END-OF-PAGE and EOP are equivalent. When the END-OF-PAGE phrase is specified, the FD entry for this file must contain a LINAGE clause. When END-OF-PAGE is specified, and the logical end of the printed page is reached during processing of the WRITE statement, the END-OF-PAGE imperative-statement is processed. The logical end of the printed page is specified in the LINAGE clause associated with record-name.

An END-OF-PAGE condition is reached when processing of a WRITE END-OF-PAGE statement causes printing or spacing within the footing area of a page body. This occurs when processing of such a WRITE statement causes the value in the LINAGE-COUNTER to equal or exceed the value specified in the WITH FOOTING phrase of the LINAGE clause. The WRITE statement is processed, and then the END-OF-PAGE imperative statement is processed.

An automatic page overflow condition is reached whenever the processing of any WRITE statement with or without the END-OF-PAGE phrase cannot be completely processed within the current page body. This occurs when a processed WRITE statement would cause the value in the LINAGE-COUNTER to exceed the number of lines for the page body specified in the LINAGE clause. In this case, the line is printed before or after the device is repositioned to the first printable line on the next logical page, as specified in the LINAGE clause.

If the END-OF-PAGE phrase is specified, the END-OF-PAGE imperative-statement is then processed. The END-OF-PAGE condition and automatic page overflow condition occur simultaneously in the following cases:

- When the WITH FOOTING phrase of the LINAGE clause is not specified. This results in no distinction between the END-OF-PAGE condition and the page overflow condition. No footing information can be printed at the bottom of a logical page when the FOOTING phrase is not specified.
- When the WITH FOOTING phrase is specified, but the processing of a WRITE statement would cause the LINAGE-COUNTER to exceed both the footing value and the page body value specified in the LINAGE clause.

E2 The keywords END-OF-PAGE and EOP are equivalent. When the END-OF-PAGE is specified, and the logical end of page is reached, during processing of the WRITE statement for the FORMATFILE file, the END-OF-PAGE imperative statement is processed. The logical end of the printed page is specified in the overflow line number parameter of the CRTPRTF command or the OVRPRTF command.

IBM Extension

F The value specified in the FORMAT phrase contains the name of the record format to use for this I-O operation. The system uses this to specify or select which record format to operate on.

The literal or identifier must be a character-string of ten characters or less. If an identifier is specified, it must be the name of one of the following:

- A Working-Storage Section entry
- A Linkage Section entry
- A record-description entry for a previously opened file.

A value of all blanks is treated as though the FORMAT phrase were not specified. If the value is not valid for the file, a FILE STATUS of 9K is returned and a USE procedure is called, if applicable for the file.

End of IBM Extension

G When the WRITE statement is processed, the system releases the record. Before the WRITE statement is processed, the user must set the key fields in the record area to the desired value.

IBM Extension

If the DUPLICATES phrase is specified, record key values for a format need not be unique (see “FILE-CONTROL Paragraph” on page 281, “RECORD KEY Clause (Indexed File)” on page 289.) In this case, the system stores the records so that later sequential access to the records allows retrieval in the order specified in DDS.

End of IBM Extension

H1 Records must be released in ascending RECORD KEY value sequence.

Note: The records must be released in ascending key sequence even though the file can be ordered in descending key sequence by a DDS option.

H2 Records can be released in any user-specified order.

I See “Indicators” on page 92.

J When the INVALID KEY condition is recognized, WRITE statement processing is unsuccessful, and the contents of the record are unaffected. See the “RECORD KEY Clause (Indexed File)” on page 289 for the order of the actions taken.

J1 An INVALID KEY condition exists when an attempt is made to write beyond the externally defined boundaries of the file.

J2 An INVALID KEY condition exists when the relative key specifies a record that already contains data.

J3 An INVALID KEY condition exists when the value of the key field in the record area equals that of an already existing record and DUPLICATES are not allowed.

J4 An INVALID KEY condition exists when the value of the key field in the record area is not greater than that for the previous record. Note that any signs on the record keys are ignored, even if the keys are defined as signed numeric in the DDS for the file.

IBM Extension

If DUPLICATES are allowed, this condition exists only if the RECORD KEY is less than that for the previous record.

End of IBM Extension

K The first record released has relative record number 1, the second has number 2, the third has number 3, and so on. If the RELATIVE KEY is specified in the file-control entry, the relative record number of the record just released is placed in the RELATIVE KEY during processing of the WRITE statement.

M The RELATIVE KEY must contain the desired relative record number for this record before the WRITE statement is issued. When the WRITE statement is processed, this record is placed at the specified relative record number position in the file, if this position is vacant.

N Required if there is more than one record format for the file.

O Optional.

P Allowed when the file is opened for I-O.

Q Allowed when the file is opened for OUTPUT.

S Allowed when the file is opened for EXTEND.

T When an attempt is made to write beyond the externally defined boundaries of the file, the processing of the WRITE statement is unsuccessful and an EXCEPTION/ERROR condition exists. The contents of record-name are unaffected. If specified, the status key is updated, and if an explicit or implicit EXCEPTION/ERROR procedure is specified for the file, the procedure is run. If no such procedure is specified, the results are unpredictable.

U The INVALID KEY phrase must be specified for files in which an applicable USE procedure is not specified.

V When end-of-volume is recognized for a multivolume OUTPUT file, the WRITE statement processes the following operations in the following order:

1. The standard ending volume label procedure is run.
2. A volume switch occurs.
3. The standard beginning volume label procedure is run.

No indication that an end-of-volume has occurred is returned to the program.

IBM Extension

Z The action of this statement can be inhibited at program run time by the INHWRT parameter of the OVRDBF CL command. When this parameter is specified, non-zero file status codes are not set for data dependent errors. Duplicate key and data conversion errors are examples of data dependent errors.

See the *CL Reference* for more information on this command.

End of IBM Extension

Arithmetic Statements

Arithmetic statements are used for computations. Individual operations are specified by the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements. The COMPUTE statement can be used to symbolically combine these operations in a formula.

Arithmetic Statement Operands

The data description of operands in an arithmetic statement need not be the same. Throughout the calculation, the compiler supplies any necessary data conversion and decimal point alignment.

Size of Operands

The maximum size of each operand is 18 decimal digits. The composite of operands (a hypothetical data item resulting from the superimposition of the operands aligned by an assumed decimal point) must not contain more than 18 decimal digits.

For the ADD and SUBTRACT statements, the composite of operands is determined by superimposing all operands in a given statement except those following the word GIVING.

For the MULTIPLY statement, the composite of operands is determined by superimposing all receiving data items.

For the DIVIDE statement, the composite of operands is determined by superimposing all receiving data items except the REMAINDER data item.

For the COMPUTE statement, the restriction on composite of operands does not apply.

For example, the items A, B, and C are defined in the Data Division as follows:

```
01 A PICTURE S9(7)V9(5).
01 B PICTURE S9(11)V99.
01 C PICTURE S9(12)V9(3).
```

If the statement ADD A, B TO C is processed, then the composite of operands for this statement consists of 17 decimal digits. It has the following implicit PICTURE clause:

```
PICTURE S9(12)V9(5)
```

IBM Extension

The composite of all operands in an arithmetic statement can have a maximum length of 30 digits.

End of IBM Extension

Overlapping Operands

When operands in an arithmetic statement share part of their storage (that is, when the operands overlap), the result of the processing of such a statement is unpredictable.

Multiple Results

When an arithmetic statement has multiple results, processing conceptually proceeds as follows:

- The statement processes all arithmetic operations to find the result to be placed in the receiving items and stores that result in a temporary location.
- A sequence of statements transfers or combines the value of this temporary result with each single receiving field. The statements are considered to be written in the same left-to-right order that the multiple results are listed.

For example, processing the following statement:

```
ADD A, B, C TO C, D(C), E.
```

is equivalent to processing the following series of statements:

```
ADD A, B, C GIVING TEMP.
ADD TEMP TO C.
ADD TEMP TO D(C).
ADD TEMP TO E.
```

`TEMP` is a compiler-supplied temporary result field. When the addition operation for `D(C)` is processed, the subscript `C` contains the new value of `C`.

Notes:

1. The compiler does not generate a temporary result field when only one identifier is specified in the following cases: in the `ADD` statement, Format 1, preceding the keyword `TO`; in the `SUBTRACT` statement, Format 1, preceding the keyword `FROM`; and in the `MULTIPLY` and `DIVIDE` statements, Format 1.
2. In all arithmetic statements, it is the user's responsibility to define data with enough digits and decimal places to ensure the desired accuracy in the final result. Refer to Appendix C, "Intermediate Result Fields" on page 537 for more information.

Common Phrases

There are several phrases common to the arithmetic statements. They are the CORRESPONDING phrase, the GIVING phrase, the ROUNDED phrase, and the SIZE ERROR phrase. Their description precedes the descriptions of the individual statements.

CORRESPONDING Phrase

The CORRESPONDING phrase allows operations to be processed on elementary items of the same name simply by specifying the group items to which they belong.

The CORRESPONDING phrase is valid in the ADD, SUBTRACT, and MOVE statements. The abbreviation CORR is equivalent to the keyword CORRESPONDING.

Both identifiers following the keyword CORRESPONDING must name group items. In this discussion, these identifiers are referred to as d1 and d2.

A pair of subordinate data items, one from d1 and one from d2, correspond if the following conditions are true:

- In an ADD or SUBTRACT statement, both of the subordinate items are elementary numeric data-items.
- In a MOVE statement, at least one of the subordinate items is elementary.
- The two subordinate items have the same name and the same qualifiers up to but not including d1 and d2.
- The subordinate items are not identified by the keyword FILLER.
- The subordinate items do not include a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause in their descriptions; if such a subordinate item is a group item, the items subordinate to it are also ignored. However, d1 and d2 themselves can contain or be subordinate to items containing a REDEFINES or OCCURS clause in their descriptions.

  For example, two data hierarchies are defined as follows:

```
05  ITEM-1 OCCURS 6 INDEXED BY X.
    10 ITEM-A ...
    10 ITEM-B ...
    10 ITEM-C REDEFINES ITEM-B ...
05  ITEM-2.
    10 ITEM-A ...
    10 ITEM-B ...
    10 ITEM-C ...
```

  If ADD CORR ITEM-2 TO ITEM-1(X) is specified, ITEM-A and ITEM-A(X) and ITEM-B and ITEM-B(X) are considered to be corresponding and are added together. ITEM-C and ITEM-C(X) are not included because ITEM-C(X) includes a REDEFINES clause in its data description. ITEM-1 is valid as either d1 or d2.

- Neither d1 nor d2 is described as a level 66, 77 or 88 item, or as a FILLER or USAGE IS INDEX item.

IBM Extension

d1 and/or d2 can be subordinate to a FILLER item.

End of IBM Extension

GIVING Phrase

If the GIVING phrase is specified, the value of the identifier that follows the word GIVING is set equal to the calculated result of the arithmetic operation. Because this identifier is not involved in the computation, it can be a numeric edited item.

ROUNDED Phrase

After decimal point alignment, the number of places in the fraction of the result of an arithmetic operation is compared with the number of places provided for the fraction of the resultant identifier.

If the size of the fractional result exceeds the number of places provided for its storage, truncation occurs unless the ROUNDED phrase is specified. When the ROUNDED phrase is specified, the least significant digit of the resultant identifier has its absolute value increased by 1 whenever the most significant digit of the excess is greater than or equal to 5.

When the resultant identifier is described by a PICTURE clause containing rightmost Ps and when the number of places in the calculated result exceeds the number of integer positions specified, rounding or truncation occurs relative to the rightmost integer position for which storage is allocated.

SIZE ERROR Phrase

A size error condition exists if, after decimal point alignment, the value of a result exceeds the largest value that can be contained in the resultant field. Division by zero or zero raised to the zero power always causes a size error condition.

In the ADD, SUBTRACT, and COMPUTE statements, the size error condition applies only to final results. In the MULTIPLY and DIVIDE statements, the size error condition applies both to final results and to intermediate results.

If the ROUNDED phrase is specified, rounding takes place before size error checking.

When a size error occurs, the subsequent action of the program depends on whether or not the SIZE ERROR phrase is specified.

If the SIZE ERROR phrase is not specified and a size error condition occurs, the value of the affected resultant identifier is unpredictable. When multiple receivers are specified, those that do not have a size error are not affected by receivers that do have the error.

If the SIZE ERROR phrase is specified and a size error condition occurs, the error results are not placed in the receiving identifier. After completion of the processing of the arithmetic operation, the imperative-statement in the SIZE ERROR phrase is processed.

If an individual arithmetic operation causes a size error condition for ADD CORRESPONDING and SUBTRACT CORRESPONDING statements, the SIZE ERROR imperative-

statement is not processed until all of the individual additions or subtraction have been completed.

ADD Statement

The ADD statement causes two or more numeric operands to be summed and the result to be stored. The formats of the ADD statement are as follows:

Format 1

```
ADD { identifier-1 } [ , identifier-2 ] . . . TO identifier-m [ ROUNDED ]
    { literal-1    } [ , literal-2    ]

    [ , identifier-n [ ROUNDED ] ] . . . [ ON SIZE ERROR imperative-statement ]
```

Format 2

```
ADD { identifier-1 } { identifier-2 } [ , identifier-3 ] . . .
    { literal-1    } { literal-2    } [ , literal-3    ]

    GIVING identifier-m [ ROUNDED ] [ , identifier-n [ ROUNDED ] ] . . .

    [ ON SIZE ERROR imperative-statement ]
```

Format 3

```
ADD { CORRESPONDING } identifier-1 TO identifier-2 [ ROUNDED ]
    { CORR          }

    [ ON SIZE ERROR imperative-statement ]
```

In Formats 1 and 2, each identifier, except those following the keyword GIVING must name an elementary numeric item. In Format 2, each identifier following the keyword GIVING must name an elementary numeric or numeric edited item. In Format 3, each identifier must name a group item. In all formats, each literal must be a numeric literal.

In Format 1, all identifiers or literals preceding the keyword TO are added together, and this sum is added to and stored immediately in identifier-m. If specified, the sum is then added to and stored immediately in identifier-n, and so on.

In Format 1, if the destination identification (after TO) is the same as the source identification (before TO), the source is modified immediately and the result is used on the remaining destination identifications.

In Format 2, at least two operands must precede the keyword GIVING. The values of these operands are added together, and the sum is stored as the new value of identifier-m, and, if specified, identifier-n, and so on.

If Format 3, elementary data items within identifier-1 are added to and stored in the corresponding elementary items within identifier-2.

For the ROUNDED and SIZE ERROR phrases, and for operand considerations, refer to the preceding "Common Phrases" on page 422 in this section.

COMPUTE Statement

The COMPUTE statement assigns the value of an arithmetic expression to one or more data items.

Format

```
COMPUTE identifier-1 [ ROUNDED ] [ , identifier-2 [ ROUNDED ] ] . . .

        = arithmetic-expression [ ON SIZE ERROR imperative-statement ]
```

The COMPUTE statement allows the user to combine arithmetic operations without the restrictions on the composite operands and/or receiving data items imposed by the rules for the ADD, SUBTRACT, MULTIPLY, and DIVIDE statements.

The identifiers that appear to the left of the equal sign (=) must name either elementary numeric items or elementary numeric edited items.

When the COMPUTE statement is processed, the value of the arithmetic expression is calculated; then this value is stored as the new value of identifier-1, identifier-2, and so on, in turn.

The arithmetic expression can be any meaningful combination of elementary numeric items, numeric literals, and arithmetic operators.

An arithmetic expression consisting of a single identifier or literal allows the user to set identifier-1, and so on, equal to the value of that identifier or literal.

For the ROUNDED and SIZE ERROR phrases, and for operand considerations, see "Common Phrases" on page 422.

DIVIDE Statement

The DIVIDE statement divides one numeric data item into others and sets the values of data items equal to the quotient and remainder. The formats of the DIVIDE statement are:

Format 1

```
DIVIDE { identifier-1 } INTO identifier-2 [ ROUNDED ]
       { literal-1    }

    [ , identifier-3 [ ROUNDED ] ] . . . [ ON SIZE ERROR imperative-statement ]
```

Format 2

```
DIVIDE { identifier-1 } { INTO } { identifier-2 } GIVING identifier-3 [ ROUNDED ]
       { literal-1    } { BY   } { literal-2    }

    [ , identifier-3 [ ROUNDED ] ] . . . [ ON SIZE ERROR imperative-statement ]
```

Format 3

```
DIVIDE { identifier-1 } { INTO } { identifier-2 } GIVING identifier-3 [ ROUNDED ]
       { literal-1    } { BY   } { literal-2    }

    REMAINDER identifer-4 [ ON SIZE ERROR imperative-statement ]
```

Each identifier except those following the keywords GIVING and REMAINDER must name an elementary numeric item. Each identifier following the keywords GIVING and REMAINDER must name an elementary numeric or numeric edited item. Each literal must be a numeric literal.

In Format 1, the value of literal-1 or identifier-1 is divided into the value of identifier-2; then the quotient is placed in identifier-2. If identifier-3 is specified, the value of literal-1 or identifier-1 is divided into identifier-3; then the quotient is placed in identifier-3, and so on.

In Format 1, if the destination identification (after INTO) is the same as the source identification (before INTO), the source is modified immediately and the result is used on the remaining destination identifications.

In Format 2, the value of identifier-1 or literal-1 is divided into/by the value of identifier-2 or literal-2. The value of the quotient is stored in identifier-3, and (if specified) identifier-4, and so on.

In Format 3, the value of identifier-1 or literal-1 is divided into/by identifier-2 or literal-2. The value of the quotient is stored in identifier-3, and the value of the remainder is stored in identifier-4.

The remainder is defined as the result of subtracting the product of the quotient and the divisor from the dividend. If identifier-3 (the quotient) is a numeric edited field, the quotient used to calculate the remainder is an intermediate field that contains the unedited quotient.

For the ROUNDED and SIZE ERROR phrases, and for operand considerations, see “Common Phrases” on page 422.

In addition to the conditions for common phrases, the following considerations apply when the ROUNDED and SIZE ERROR phrases are used in Format 3.

- When the ROUNDED phrase is specified, the quotient used to calculate the remainder is an intermediate field which contains the quotient truncated rather than rounded.
- When the ON SIZE ERROR phrase is specified and the size error condition occurs on the quotient, no remainder calculation is meaningful. Therefore, the con-

tents of the quotient field (identifier-3) and the remainder field (identifier-4) are unchanged.

- When the ON SIZE ERROR phrase is specified and the size error occurs on the remainder, the contents of the remainder field (identifier-4) are unchanged.

Note: In the two preceding cases, the user must analyze the results to determine which situation has actually occurred.

MULTIPLY Statement

The MULTIPLY statement causes numeric items to be multiplied and sets the values of data items equal to the results. The formats of the MULTIPLY statement are:

Format 1

```
MULTIPLY { identifier-1 } BY identifier-2 [ ROUNDED ]
         { literal-1    }
    [ , identifier-3 [ ROUNDED ] ] . . . [ ON SIZE ERROR imperative-statement ]
```

Format 2

```
MULTIPLY { identifier-1 } BY { identifier-2 } GIVING identifier-3 [ ROUNDED ]
         { literal-1    }    { literal-2    }
    [ , identifier-4 [ ROUNDED ] ] . . . [ ON SIZE ERROR imperative-statement ]
```

Each identifier except those following the keyword GIVING must name an elementary numeric item. Each identifier following the keyword GIVING must name an elementary numeric or numeric edited item. Each literal must be a numeric literal.

In Format 1, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2; the product is then placed in identifier-2. If identifier-3 is specified, the value of identifier-1 or literal-1 is multiplied by the value of identifier-3; the product is then placed in identifier-3, and so on.

In Format 1, if the destination identification (after BY) is the same as the source identification (before BY), the source is modified immediately and the result is used on the remaining destination identifications.

In Format 2, the value of identifier-1 or literal-1 is multiplied by the value of identifier-2 or literal-2; the product is then stored in identifier-3, and, if specified, identifier-4, and so on.

For the ROUNDED and SIZE ERROR phrases, and for operand considerations, see “Common Phrases” on page 422.

SUBTRACT Statement

The SUBTRACT statement causes either one, or the sum of two or more numeric items to be subtracted from one or more numeric items and the result to be stored. The formats of the SUBTRACT statement are:

Format 1

```
SUBTRACT { identifier-1 } [ , identifier-2 ] . . . FROM identifier-3 [ ROUNDED ]
         { literal-1    }  [ , literal-2    ]

   [ , identifier-4 [ ROUNDED ] ] . . . [ ON SIZE ERROR imperative-statement ]
```

Format 2

```
SUBTRACT { identifier-1 } [ , identifier-2 ] . . . FROM { identifier-3 }
         { literal-1    }  [ , literal-2    ]          { literal-3    }

    GIVING identifier-4 [ ROUNDED ] [ , identifier-5 [ ROUNDED ] ] . . .

    [ ON SIZE ERROR imperative-statement ]
```

Format 3

```
SUBTRACT { CORRESPONDING } identifier-1 FROM identifier-2 [ ROUNDED ]
         { CORR          }

    [ ON SIZE ERROR imperative-statement ]
```

In Formats 1 and 2, each identifier except those following the keyword GIVING must name an elementary numeric item. In Format 2, each identifier following the keyword GIVING must name a numeric elementary or numeric edited elementary item. In Format 3, each identifier must name a group item. In all formats, each literal must be a numeric literal.

In Format 1, all identifiers or literals preceding the keyword FROM are added together, and this sum is subtracted from and stored immediately in identifier-3, and then, if specified, subtracted from and stored immediately in identifier-4, and so on.

In Format 1, if the destination identification (after FROM) is the same as one of the identifications (before FROM), the source is modified immediately and the result is used on the remaining destination identifications.

In Format 2, all identifiers or literals preceding the keyword FROM are added together and this sum is subtracted from identifier-3 or literal-3. The result of the subtraction is stored as the new value of identifier-4, and, if specified, identifier-5, and so on.

In Format 3, elementary data items within identifier-1 are subtracted from and stored in the corresponding elementary data items within identifier-2.

For the ROUNDED and SIZE ERROR phrases, and for operand considerations, see "Common Phrases" on page 422.

Data Manipulation Statements

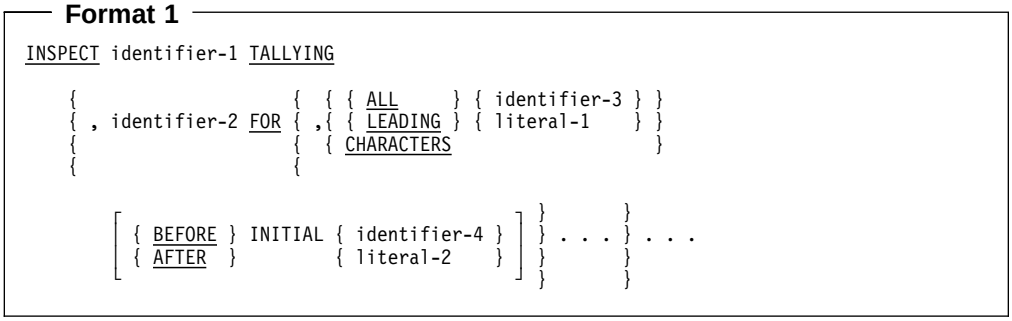
Movement and inspection of data are the functions of the following COBOL statements: INSPECT, MOVE, STRING, and UNSTRING.

When the sending and receiving fields of a data manipulation statement share a part of their storage (that is, when the operands overlap), the result of the processing of such a statement is unpredictable.

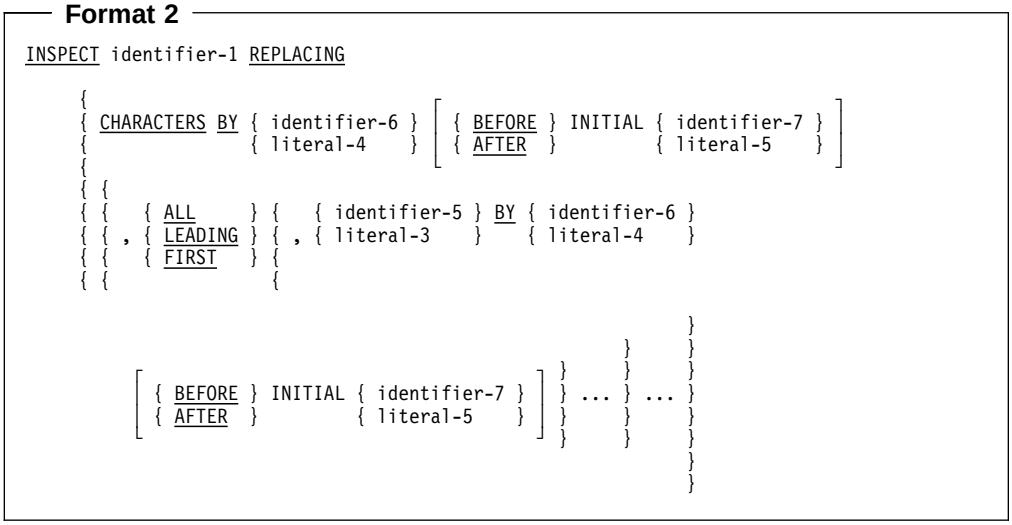
INSPECT Statement

The INSPECT statement specifies that characters in a data item are to be counted, replaced, or counted and replaced. The formats of the INSPECT statement are:

Format 1



Format 2



Format 3

```
INSPECT identifier-1 TALLYING

    {
    {                    {
    { , identifier-2 FOR {   { { ALL     } { identifier-3 } }
    {                    { , { { LEADING } { literal-1    } }
    {                    {   { CHARACTERS                   }
    {
    {

                                                          }
                                                          }
    [ { BEFORE } INITIAL { identifier-4 } ] }             } . . .
    [ { AFTER  }         { literal-2    } ] }             }
                                          }               }
                                                          }

    REPLACING

     {                                                                  ]  }
     { CHARACTERS BY { identifier-6 } [ { BEFORE }   INITIAL { identifier-7 } ]  }
     {               { literal-4    } [ { AFTER  }           { literal-5    } ]  }
     {                                                                       }
     { {                {                                                    }
     { {   { ALL     } {   { identifier-5 } BY { identifier-6 }              }
     { { , { LEADING } { , { literal-3    }    { literal-4    }              }
     { {   { FIRST   } {                                                    }
     {                                                                      }
     {    [                                          ] }     }             }
     {    [ { BEFORE } INITIAL { identifier-7 } ] } ... } ...             }
     {    [ { AFTER  }         { literal-5    } ] }     }                  }
     {                                          }     }                  }
     {                                                                      }
     {                                                                      }
```

Either the TALLYING or the REPLACING phrase must be specified. Both the TALLYING and REPLACING phrases can be specified. If both TALLYING and REPLACING are specified (Format 3), all tallying is processed before any replacement is made.

Identifier-1 is the inspected item. Identifier-1 must be an elementary or group item with USAGE DISPLAY.

All other identifiers except identifier-2 (the count field) must be elementary alphabetic, alphanumeric, or zoned decimal items. Each is treated according to its data category. Each data category is treated as follows:

- Alphabetic or alphanumeric items are treated as a character-string.
- Alphanumeric edited, numeric edited, or unsigned numeric (zoned decimal) items are treated as though redefined as alphanumeric, and the INSPECT statement refers to the alphanumeric item.
- Signed numeric (zoned decimal) items are treated as though moved to an unsigned zoned decimal item of the same length, and then treated as though redefined as alphanumeric. The INSPECT statement refers to the alphanumeric item.

Each literal must be nonnumeric and can be any figurative constant except ALL.

The comparison operands of the TALLYING phrase (literal-1 or identifier-3, and so on) and/or REPLACING phrase (literal-3 or identifier-5, and so on) are compared in the left-to-right order specified in the INSPECT statement. A maximum of 15 comparison operands may be specified for each REPLACING and each TALLYING

phrase. When the INSPECT statement is contained within IF statements, this maximum number is reduced by the number of nested IF statements.

When the TALLYING/REPLACING operands are the compared operands, the following comparison rules apply:

1. When both the TALLYING and REPLACING phrases are specified, the INSPECT statement is processed as if an INSPECT TALLYING statement were specified and immediately followed by an INSPECT REPLACING statement.
2. The first operand is compared with an equal number of leftmost contiguous characters in the inspected item. The operand matches the inspected characters only if both are equal, character for character.
3. If no match occurs for the first operand, the comparison is repeated for each successive operand until either a match is found or all operands have been acted upon.
4. If a match is found, tallying or replacing takes place as described in TALLYING/REPLACING phrase descriptions. In the inspected item, the first character following the rightmost matching character is now considered the leftmost character position. The process described in comparison rules 2 and 3 is then repeated.
5. If no match is found, the first character in the inspected item following the leftmost inspected character is now considered the leftmost character position. The process described in comparison rules 2 and 3 is then repeated.
6. If the CHARACTERS phrase is specified, an implied 1-character operand is used in the process described in rules 2 and 3. The implied character is considered to always match the inspected character of the item inspected.
7. The actions taken in comparison rules 1 through 6–which are defined as the comparison cycle–are repeated until the rightmost character in the inspected item has either been matched or has been considered as the leftmost character position. Inspection then terminates.

Figure 81 on page 432 illustrates INSPECT statement comparisons.

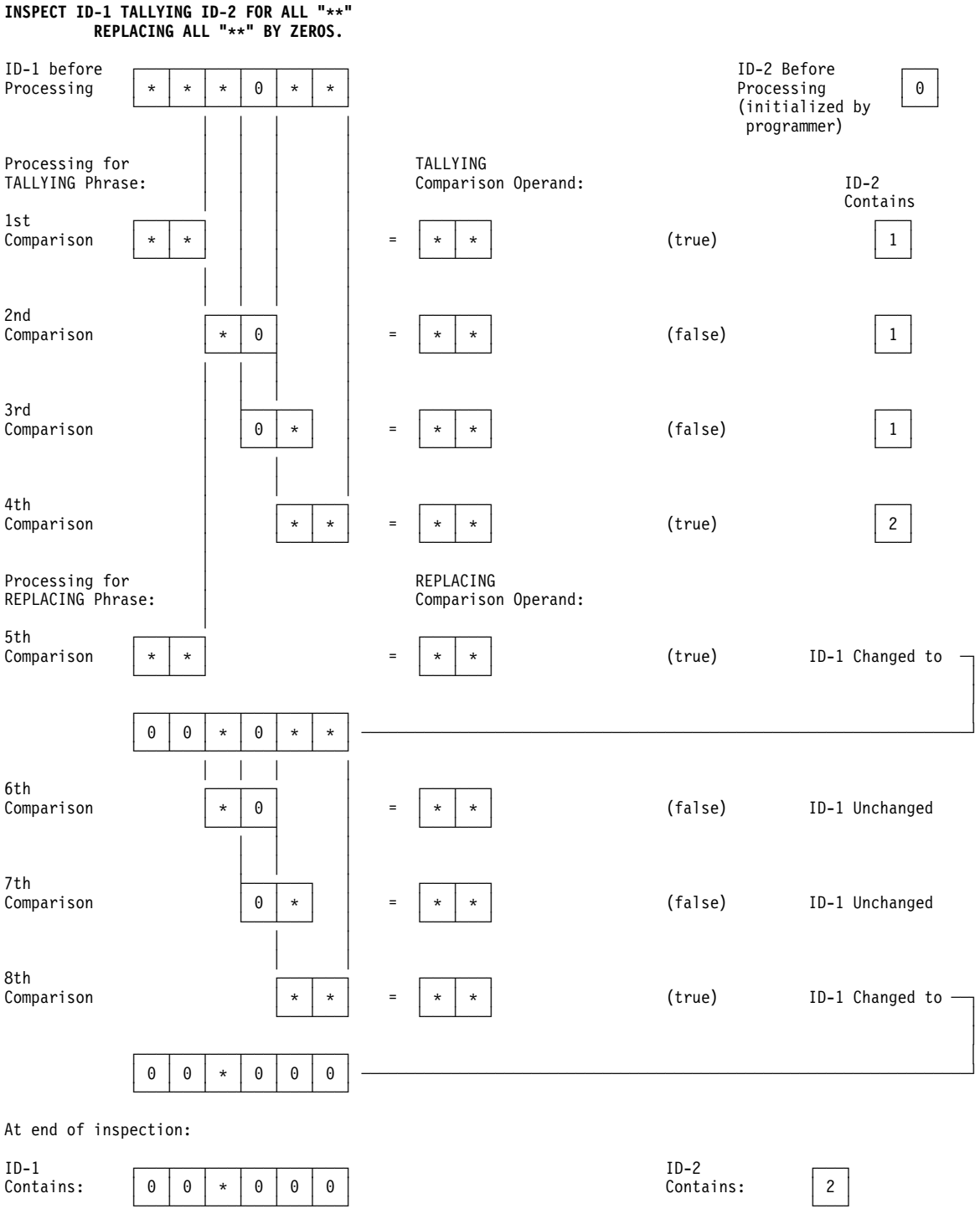


Figure 81. INSPECT Statement Processing Results

Note: When the BEFORE/AFTER phrase is specified, the preceding results are modified as described in the BEFORE/AFTER phrase description.

INSPECT

The following example shows an INSPECT statement.

```
.. 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7

  DATA DIVISION.
  WORKING-STORAGE SECTION.
  01  ID-1      PIC X(10)   VALUE "ACADEMIANS".
  01  CONTR-1    PIC 99     VALUE 00.
  01  CONTR-2    PIC 99     VALUE ZEROS.
  PROCEDURE DIVISION.
 *    THIS ILLUSTRATES AN INSPECT STATEMENT WITH 2 VARIABLES.
  100-BEGIN-PROCESSING.
      DISPLAY CONTR-1 SPACE CONTR-2.
  101-MAINLINE-PROCESSING.
      PERFORM COUNT-IT THRU COUNT-EXIT.
      STOP RUN.
  COUNT-IT.
      INSPECT ID-1
          TALLYING CONTR-1
              FOR CHARACTERS BEFORE INITIAL "AD"
              CONTR-2
              FOR ALL "MIANS".
  DISPLAY-COUNTS.
      DISPLAY "CONTR-1 = " CONTR-1.
      DISPLAY "CONTR-2 = " CONTR-2.
      DISPLAY "*********EOJ*********".
  COUNT-EXIT.
      EXIT.
```

Resultant Output:

```
      00 00
      CONTR-1 = 02
      CONTR-2 = 01
      *********EOJ*********
```

TALLYING Phrase

Identifier-2 is the tallying field and must be an elementary integer item defined without the symbol P in its PICTURE character-string. Identifier-2 must be initialized before the INSPECT statement is processed.

Identifier-3 or literal-1 is the comparison operand. If the comparison operand is a figurative constant, it is considered to be a one-character nonnumeric literal.

REPLACING Phrase

Identifier-5 or literal-3 is the comparison operand. Identifier-6 or literal-4 is the replacement field.

The comparison operand and the replacement field must be the same length. The following replacement rules apply:

- If the comparison operand is a figurative constant, it is considered to be a one-character nonnumeric literal. Each character in the inspected item equivalent to the figurative constant is replaced by the single-character replacement field, which must be one character in length.

- If the replacement field is a figurative constant, it is considered to be the same length as the comparison operand. Each nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the replacement field.
- When the comparison operand and replacement fields are character-strings, each nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the character-string specified in the replacement field.
- Once replacement has occurred in a given character position in the inspected item, no further replacement for that character position is made in this processing of the INSPECT statement.

BEFORE/AFTER Phrases

The keywords BEFORE and AFTER should not be used in the same statement.

When either of these phrases is specified, the preceding actions for tallying and replacing are modified.

Identifier-4, identifier-7, literal-2, and literal-5 are delimiters. Tallying and/or replacement of the inspected item is bounded by their presence; however, the delimiters themselves are not counted or replaced.

If the delimiter (literal-2 or literal-5) is a figurative constant, it is considered to be one character in length.

In the REPLACING phrase, if the CHARACTERS phrase is specified, the delimiter (literal-5 or identifier-7) must be one character in length.

When the BEFORE phrase is specified, tallying and/or replacement of the inspected item begins at the leftmost character and continues until the first occurrence of the delimiter is encountered. If no delimiter is present in the inspected item, tallying and/or replacement continues to the rightmost character.

When the AFTER phrase is specified, tallying and/or replacement of the inspected item begins with the first character to the right of the delimiter and continues to the rightmost character in the inspected item. If no delimiter is present in the inspected item, no tallying or replacement takes place.

When the BEFORE/AFTER phrase is not specified, the following actions take place when the INSPECT TALLYING statement is processed:

- If the ALL phrase is specified, the tallying field is increased by one for each nonoverlapping occurrence in the inspected item of the comparison operand. This process begins at the leftmost character position and continues to the rightmost.
- If the LEADING phrase is specified, the tallying field is increased by one for each contiguous nonoverlapping occurrence of the comparison operand in the inspected item, provided the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which the comparison operand is eligible to participate.
- If the CHARACTERS phrase is specified, the tallying field is increased by one for each character (including the space character) in the inspected item. Thus, processing of the INSPECT TALLYING statement increases the value in the tallying field by the number of characters in the inspected item.

When the BEFORE/AFTER phrase is not specified, the following actions take place when the INSPECT REPLACING statement is processed:

- If the CHARACTERS phrase is specified, the replacement field must be 1 character in length. Each character in the inspected field is replaced by the replacement field. This process begins at the leftmost character and continues to the rightmost.
- If the ALL phrase is specified, each nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the replacement field, beginning at the leftmost character and continuing to the rightmost.
- If the LEADING phrase is specified, each contiguous nonoverlapping occurrence of the comparison operand in the inspected item is replaced by the replacement field, provided that the leftmost such occurrence is at the point where comparison began in the first comparison cycle for which this replacement field is eligible to participate.
- If the FIRST phrase is specified, the leftmost occurrence of the comparison operand in the inspected item is replaced by the replacement field.

INSPECT Statement Examples

The following examples illustrate some uses of the INSPECT statement. In all instances, the programmer has initialized the COUNTR field to zero before the INSPECT statement is processed.

```
INSPECT ID-1 REPLACING CHARACTERS BY ZERO.
```

ID-1 Before	COUNTR After	ID-1 After
1234567	0	0000000
HIJKLMN	0	0000000

```
INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS REPLACING CHARACTERS BY SPACES.
```

ID-1 Before	COUNTR After	ID-1 After
1234567	7	
HIJKLMN	7	

```
INSPECT ID-1 REPLACING CHARACTERS BY ZEROS BEFORE INITIAL QUOTE.
```

ID-1 Before	COUNTR After	ID-1 After
456"ABEL	0	000"ABEL
ANDES"12	0	00000"12
"TWAS BR	0	"TWAS BR

```
INSPECT ID-1 TALLYING COUNTR FOR CHARACTERS AFTER INITIAL "S" REPLACING ALL
"A" BY "O".
```

ID-1 Before	COUNTR After	ID-1 After
ANSELM	3	ONSELM
SACKET	5	SOCKET
PASSED	3	POSSED

```
INSPECT ID-1 TALLYING COUNTR FOR LEADING "0" REPLACING FIRST "A" BY "2"
AFTER INITIAL "C".
```

ID-1 Before	COUNTR After	ID-1 After
00ACADEMY00	2	00AC2DEMY00
0000ALABAMA	4	0000ALABAMA
CHATAM0000	0	CH2THAM0000

Note: The INSPECT statement is useful for filling all or part of a data item with spaces or zeros. It is also useful for counting the number of times a specific character (for example, zero, space, asterisk) occurs in a data item. In addition, it can be used to translate characters from one collating sequence to another.

MOVE Statement

The MOVE statement transfers data from one area of storage to one or more other areas. The formats of the MOVE statement are as follows:

Format 1

```
MOVE { identifier-1 } TO identifier-2 [ , identifier-3 ] . . .
     { literal      }
```

Format 2

```
MOVE { CORRESPONDING } identifier-1 TO identifier-2
     { CORR          }
```

General Considerations

Identifier-1 or literal is the sending area. Identifier-2, identifier-3, and so on are the receiving areas.

An index data item cannot be specified in a MOVE statement. Any subscripting or indexing associated with the sending item is evaluated only once: immediately before the data is moved to the first receiving field. Any subscripting or indexing associated with the receiving items is evaluated immediately before the data is moved into the receiving field.

For example, the result of the statement:

```
MOVE A (B) TO B, C (B).
```

is equivalent to

```
MOVE A (B) TO TEMP.
MOVE TEMP TO B.
MOVE TEMP TO C (B).
```

where TEMP has been defined as an intermediate result item. The subscript B changed in value between the time the first move took place and the time the final move to C (B) was processed.

After processing of a MOVE statement, a sending field contains the same data as before processing, unless a receiving field overlaps the sending field.

Unexpected results can occur when a redefining item is moved to the redefined item (that is, if B REDEFINES C and the statement MOVE B TO C is processed). Unexpected results can also occur when a redefined item is moved to an item redefining it (from the previous example, unexpected results occur if the statement MOVE C TO B is processed).

Elementary Moves

An elementary move is one in which both the sending and receiving items are elementary items. Each elementary item belongs to one of the following categories:

- Numeric: Includes numeric data items, numeric literals, and the figurative constant ZERO/ZEROS/ZEROES
- Alphabetic: Includes alphabetic data items and the figurative constant SPACE/SPACES.

Both identifiers following the keyword CORRESPONDING must name group items. In this discussion, these identifiers are referred to as d1 and d2.

A pair of subordinate data items, one from d1 and one from d2, correspond if the following conditions are true:

- At least one of the subordinate items is elementary.
- The two subordinate items have the same name and the same qualifiers up to but not including d1 and d2.
- The subordinate items are not identified by the keyword FILLER.
- The subordinate items do not include a REDEFINES, RENAMES, OCCURS, or USAGE IS INDEX clause in their descriptions; if such a subordinate item is a group item, the items subordinate to it are also ignored. However, d1 and d2 themselves can contain or be subordinate to items containing a REDEFINES or OCCURS clause in their description.

  For example, two data hierarchies are defined as follows:

```
05 ITEM-1 OCCURS 6 INDEXED BY X.
   10 ITEM-A ...
   10 ITEM-B ...
   10 ITEM-C REDEFINES ITEM-B ...
05 ITEM-2.
   10 ITEM-A ...
   10 ITEM-B ...
   10 ITEM-C ...
```

If MOVE CORR ITEM-2 TO ITEM-1(X) is specified, ITEM-A and ITEM-A(X), and ITEM-B and ITEM-B(X) are considered to be corresponding and the moves are processed. ITEM-C and ITEM-C(X) are not included because ITEM-C(X) includes a REDEFINES clause in its data description. ITEM-1 is valid as either d1 or d2.

- Neither d1 nor d2 is described as a level 66, 77 or 88 item, or as a FILLER or USAGE IS INDEX item.

IBM Extension

d1 and/or d2 can be subordinate to a FILLER item.

End of IBM Extension

- Alphanumeric: Includes alphanumeric data items, nonnumeric literals, and all figurative constants except ZERO and SPACE.
- Alphanumeric edited: Includes alphanumeric edited data items.
- Numeric edited: Includes numeric edited data items.

IBM Extension

- Boolean: Includes Boolean data items and Boolean literals.

End of IBM Extension

Valid elementary moves are processed according to the following rules:

- Any necessary conversion of data from one form of internal representation to another along with any specified editing in the receiving item takes place during the move.
- For an alphanumeric or alphanumeric edited receiving item:
  - Justification and any necessary space filling take place as described under "Standard Alignment Rules" on page 313. Unused character positions are filled with spaces.
  - If the size of the sending item is greater than the size of the receiving item, excess characters at the right are truncated after the receiving item is filled.
  - If the sending item has an operational sign, the absolute value is used. If the operational sign occupies a separate character, that character is not moved, and the size of the sending item is considered to be one less than its actual size.
- For a numeric or numeric edited receiving item:
  - Alignment by decimal point and any necessary zero filling take place as described under "Standard Alignment Rules" on page 313, except where zeros are replaced because of editing requirements.
  - If the receiving item is signed, the sign of the sending item is placed in the receiving item, with any necessary sign conversion. If the sending item is unsigned, a positive operational sign is generated for the receiving item.
  - The absolute value of the sending item is used if the receiving item has no operational sign.
  - If the sending item has more digits to the left or right of the decimal point than the receiving item can contain, excess digits are truncated.
  - When the sending item is alphanumeric, the data is moved as if the sending item were described as an unsigned integer. It is the user's responsibility to ensure that the data is numeric.
- For an alphabetic receiving field:
  - Justification and any necessary space filling take place as described under "Standard Alignment Rules" on page 313

– If the size of the sending item is greater than the size of the receiving item, excess characters at the right are truncated after the receiving item is filled.

IBM Extension

- For a Boolean receiving field, only the first byte of the sending item is moved.

End of IBM Extension

Note: If the receiving field is alphanumeric or numeric edited, and the sending field is a scaled integer (that is, it has a `P` as the rightmost character in its `PICTURE` character-string), the scaling positions are treated as trailing zeros when the `MOVE` statement is processed.

Figure 82 shows valid and invalid elementary moves for each category.

Group Moves

A group move is one in which one or both of the sending and receiving fields are a group item. A group move is treated exactly as though it were an alphanumeric elementary move except that data is not converted from one form of internal representation to another. In a group move, the receiving area is filled without consideration for the individual elementary items contained within either the sending area or the receiving area. See “OCCURS Clause” on page 476 for additional information.

Sending Item Category	Receiving Item Category						
	Alphabetic	Alphanumeric	Alphanumeric Edited	Numeric Integer	Numeric Noninteger	Numeric Edited	Boolean
Alphabetic and SPACE	YES	YES	YES	NO	NO	NO	NO
Alphanumeric	YES	YES	YES	YES[4]	YES[4]	YES[4]	YES[5]
Nonnumeric Literal	YES	YES	YES	YES[1]	YES[1]	YES[1]	YES[5]
Alphanumeric Edited	YES	YES	YES	NO	NO	NO	NO
Numeric Integer[2]	NO	YES	YES	YES	YES	YES	NO
Numeric Noninteger[2]	NO	NO	NO	YES	YES	YES	NO
Numeric Edited	NO	YES	YES	NO	NO	NO	NO
LOW/HIGH-VALUES QUOTES	NO	YES	YES	NO	NO	NO	NO
ZERO	NO	YES	YES	YES	YES	YES	YES
Boolean[3]	NO	YES	YES	NO	NO	NO	YES

YES = move is valid
NO = move is invalid

1 Moved only if an unsigned integer.
2 Includes numeric literals.
3 Includes Boolean literals.
4 Compiler assumes alphanumeric item is an unsigned integer.
5 Compiler assumes item is a 0 or a 1.

Figure 82. Valid and Invalid Elementary Moves

Note: Boolean in the Sending Item Category and in the Receiving Item Category represents an IBM Extension to ANS COBOL X3.23-1974.

Format 1 Considerations

When Format 1 is specified, the identifiers can be either group or elementary items. The data in the sending area is moved into the first receiving area (identifier-2); then it is moved from the sending area into the second receiving area (identifier-3), and so on.

Format 2 Considerations

CORRESPONDING Phrase: The CORRESPONDING phrase allows data to be moved between elementary items of the same name simply by specifying the group items to which they belong.

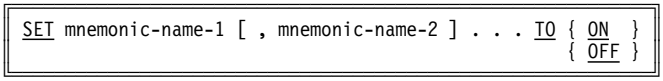
The abbreviation CORR is equivalent to the keyword CORRESPONDING.

SET Statement

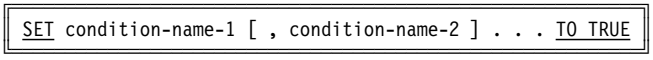
IBM Extension

The SET statement is used to alter the status of external switches and the values of conditional variables. See "SET Statement" on page 488 for the Format 3 and Format 4.

Format 1



Format 2



For Format 1 each mnemonic-name must be associated with an external switch, the status of which can be altered. The only external switches allowed are the UPSI switches, UPSI-0 through UPSI-7.

The status of each external switch associated with the specified mnemonic-name is modified such that the truth value resultant from evaluation of a condition-name associated with that switch will reflect an on status if the ON phrase is specified, or an off status if the OFF phrase is specified. For additional information, refer to "Switch-Status Condition" on page 359 earlier in this chapter.

Format 2 allows conditional items to be set to their stated values. The literal in the VALUE clause associated with the condition-name is moved to the conditional variable according to the rules for elementary moves.

If multiple condition-names are specified, the results are the same as those obtained if a separate SET statement were written for each condition-name in the same order specified in the SET statement.

End of IBM Extension

STRING Statement

The STRING statement gives the programmer the ability to concatenate the partial or complete contents of two or more data items into a single data item.

Format

```
STRING { identifier-1 } [ , identifier-2 ] . . . DELIMITED BY { identifier-3 }
       { literal-1    } [ , literal-2    ]                    { literal-3    }
                                                              { SIZE         }

    [ , { identifier-4 } [ , identifier-5 ] . . . DELIMITED BY { identifier-6 } ]
      { literal-4    } [ , literal-5    ]                    { literal-6    }   . . .
                                                             { SIZE         }

    INTO identifier-7 [ WITH POINTER identifier-8 ]

    [ ON OVERFLOW imperative-statement ]
```

Each literal must be a nonnumeric literal; each can be any figurative constant except the ALL literal. When a figurative constant is specified, it is considered a one-character nonnumeric literal.

All identifiers except identifier-8 (the pointer item) must have USAGE DISPLAY, explicitly or implicitly.

The sending fields are identifier-1, identifier-2, identifier-4, identifier-5, or their corresponding literals.

The receiving field is identifier-7, which must be an elementary alphanumeric item without editing symbols and without the JUSTIFIED clause in its description.

The delimiters are identifier-3, identifier-6, or their corresponding literals, or the keyword SIZE. The delimiters specify the character(s) delimiting the data to be transferred; when SIZE is specified, the complete sending area is transferred.

When the sending field or any of the delimiters are elementary numeric items, they must be described as integers, and their PICTURE character-strings must not contain the symbol P.

The pointer field is identifier-8, which must be an elementary integer data item large enough to contain a value equal to the length of the receiving area plus one. The pointer field must not contain the symbol P in its PICTURE character-string.

STRING Statement Processing

When the STRING statement is processed, data is transferred from the sending fields to the receiving field. The order in which sending fields are processed is the order in which they are specified. The following rules apply:

- Characters from the sending fields are transferred to the receiving field according to the rules for alphanumeric to alphanumeric elementary moves except that no space filling is provided.
- When the DELIMITED BY identifier/literal is specified, the contents of each sending item are transferred character by character beginning with the leftmost and continuing until either a delimiter for this sending field is reached (the delimiter itself is not transferred) or the rightmost character of this sending field has been transferred.
- When DELIMITED BY SIZE is specified, each sending field is transferred in its entirety to the receiving field.
- When the receiving field is filled or when all the sending fields have been processed, the operation is ended.
- When the POINTER phrase is specified, an explicit pointer field is available to the COBOL user to control placement of data in the receiving field. The user must set the explicit pointer's initial value, which must not be less than one and not more than the character count of the receiving field. The pointer field must be defined as large enough to contain a value equal to the length of the receiving field plus 1; this precludes arithmetic overflow when the system updates the pointer at the end of the transfer.
- When the POINTER phrase is not specified, no pointer is available to the user. However, an implicit pointer with an initial value of one is used by the system.
- When the STRING statement is processed, the initial pointer value (explicit or implicit) points to the first character position within the receiving field into which data is to be transferred. Beginning at that position, data is then positioned character by character from left to right. After each character is positioned, the explicit or implicit pointer is incremented by one. The value in the pointer field is changed only in this manner. At the end of processing, the pointer value always indicates one character beyond the last character transferred into the receiving field.
- If, at any time during or after initiation of STRING statement processing, the pointer value (explicit or implicit) is less than one or exceeds a value equal to the length of the receiving field, no more data is transferred into the receiving field and, if specified, the ON OVERFLOW imperative-statement is processed. (The ON OVERFLOW statement is not processed unless there was an attempt to move in one or more characters beyond the end of identifier-7.)
- If the ON OVERFLOW phrase is not specified, then when the preceding conditions occur, control passes to the next executable statement.

After STRING statement processing is completed, only that part of the receiving field into which data was transferred is changed. The rest of the receiving field contains the data that was present before this processing of the STRING statement.
Figure 83 on page 443 illustrates the rules of processing for the STRING statement.

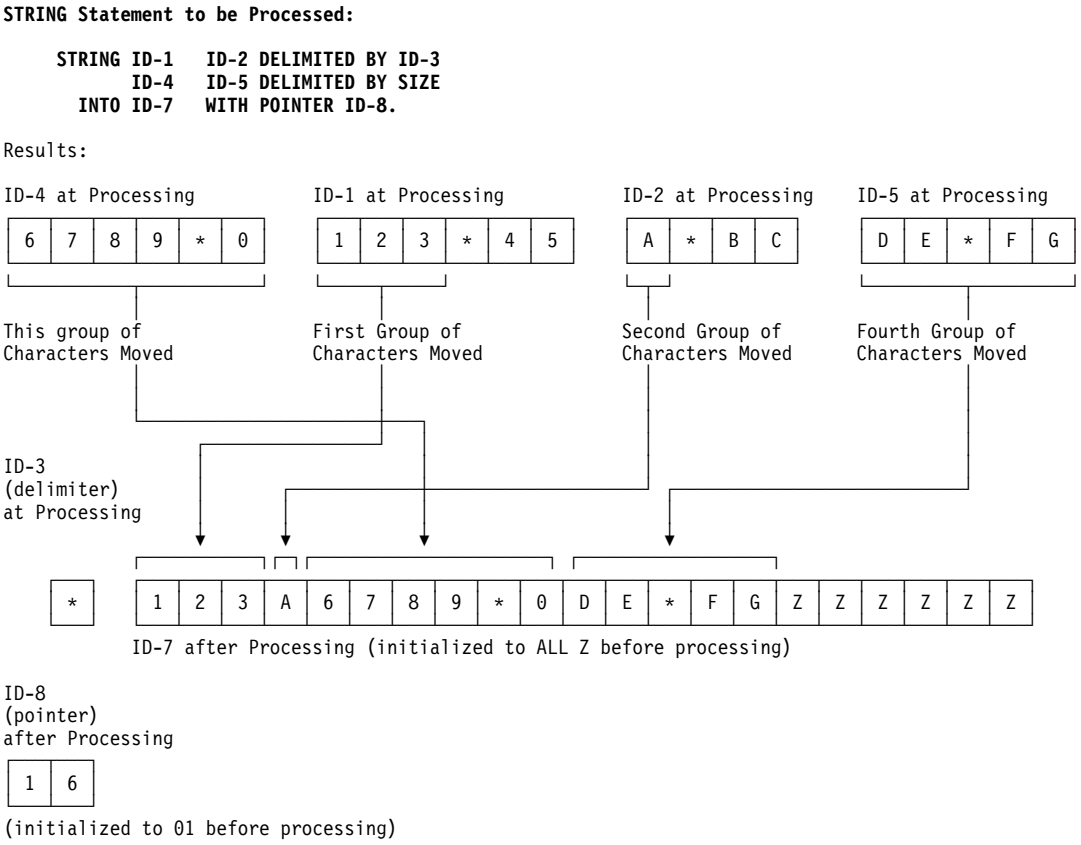


Figure 83. STRING Statement Processing Results

STRING Statement Example

The following example illustrates some of the considerations that apply to the STRING statement.

In the Data Division, the programmer has defined the following fields:

```
01 RPT-LINE     PICTURE X(120).
01 LINE-POS     PICTURE 99.
01 LINE-NO      PICTURE 9(5) VALUE 1.
01 DEC-POINT    PICTURE X VALUE ".".
```

In the File Section, he has defined the following input record:

```
01 RCD-01.
   05 CUST-INFO.
      10 CUST-NAME  PICTURE X(15).
      10 CUST-ADDR  PICTURE X(34).
   05 BILL-INFO.
      10 INV-NO     PICTURE X(6).
      10 INV-AMT    PICTURE $$,$$$.99.
      10 AMT-PAID   PICTURE $$,$$$.99.
      10 DATE-PAID  PICTURE X(8).
      10 BAL-DUE    PICTURE $$,$$$.99.
      10 DATE-DUE   PICTURE X(8).
```

The user wants to construct an output line consisting of portions of the information from RCD-01. The line is to consist of a line number, customer name and address, invoice number, date due, and balance due, truncated to the dollar figure shown.

The record as read in contains the following information:

```
J.B.bSMITHbbbbb
444bSPRINGbST.,bCHICAGO,bILL.bbbbb
A14275
$4,736.85
$2,400.00
09/22/76
$2,336.85
10/22/76
```

In the Procedure Division, the user initializes RPT-LINE to SPACES and sets LINE-POS (which is to be used as the pointer field) to 4. Then he issues this STRING statement:

```
STRING LINE-NO SPACE
    CUST-INFO SPACE
    INV-NO SPACE
    DATE-DUE SPACE
  DELIMITED BY SIZE,
    BAL-DUE
  DELIMITED BY DEC-POINT
  INTO RPT-LINE
  WITH POINTER LINE-POS.
```

When the statement is processed, the following actions take place:

1. The field LINE-NO is moved into positions 4 through 8 of RPT-LINE.
2. A space is moved into position 9.
3. The group item CUST-INFO is moved into positions 10 through 58.
4. A space is moved into position 59.
5. INV-NO is moved into positions 60 through 65.
6. A space is moved into position 66.
7. DATE-DUE is moved into positions 67 through 74.
8. A space is moved into position 75.
9. The portion of BAL-DUE that precedes the decimal point is moved into positions 76 through 81.

After the STRING statement has been processed:

- RPT-LINE appears as shown in Figure 84 on page 445.
- LINE-POS contains the value 82.

Note: One STRING statement can be written instead of a series of MOVE statements.

```
Column
 4     10              25
 |     |               |
 |     |               |
 00001 J.B. SMITH      444 SPRING ST., CHICAGO, ILL.

 60      67        76
 |       |         |
 |       |         |
 A14725  10/22/76  $2,336
```

Figure 84. STRING Statement Example Output Data

UNSTRING Statement

The UNSTRING statement causes contiguous data in a sending field to be separated and placed into multiple receiving fields.

Format

```
UNSTRING identifier-1

    [ DELIMITED BY [ ALL ] { identifier-2 } [ , OR [ ALL ] { identifier-3 } ] . . . ]
                           { literal-1    }               { literal-2    }

    INTO identifier-4 [ , DELIMITER IN identifier-5 ] [ , COUNT IN identifier-6 ]

       [ , identifier-7 [ , DELIMITER IN identifier-8 ] [ , COUNT IN identifier-9 ] ] . . .

    [ WITH POINTER identifier-10 ] [ TALLYING IN identifier-11 ]

    [ ON OVERFLOW imperative-statement ]
```

Each literal must be a nonnumeric literal; each may be any figurative constant except ALL literal. When a figurative constant is specified, it is considered to be a one-character nonnumeric literal.

Sending Field

Identifier-1 is the sending field. It must be an alphanumeric data item. Data is transferred from this field to the receiving fields.

DELIMITED BY Phrase: This phrase specifies delimiters within identifier-1 that control the data transfer.

The delimiters are identifier-2, identifier-3, or their corresponding literals. Each identifier or literal specified represents one delimiter. No more than 30 delimiters can be specified. Each must be an alphanumeric data item.

If a delimiter contains two or more characters, it is recognized in the sending field only if the delimiter characters are contiguous and, in the sequence specified, in the delimiter item.

When two or more delimiters are specified, an OR condition exists and each nonoverlapping occurrence of any one of the delimiters is recognized in the sending field in the sequence specified. For example, if DELIMITED BY AB OR BC is specified, then an occurrence of either AB or BC in the sending field is considered a delimiter. An occurrence of ABC is considered an occurrence of AB, and the search for another delimiter resumes with C.

When the DELIMITED BY ALL phrase is not specified, and two or more contiguous occurrences of any delimiter are encountered, the current data receiving field is filled with spaces or zeros according to the description of the data receiving field.

When the DELIMITED BY ALL phrase is specified, one or more contiguous occurrences of any delimiter are treated as if they were only one occurrence, and this one occurrence is moved to the delimiter receiving field (if specified). The delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved into the current delimiter receiving field according to the rules of the MOVE statement.

The DELIMITER IN and COUNT IN phrases can be specified only if the DELIMITER BY phrase is specified.

Data Receiving Fields

Identifier-4, identifier-7, and so on, are the data receiving fields and must have USAGE DISPLAY. These fields can be defined as:

- Alphabetic (without the symbol B in the PICTURE string)
- Alphanumeric
- Numeric (without the symbol P in the PICTURE string).

These fields must not be defined as alphanumeric edited or numeric edited items. Data is transferred to these fields from the sending field.

DELIMITER IN Phrase: The delimiter receiving fields are identifier-5, identifier-8, and so on. These identifiers must be alphanumeric. They must not be defined as alphanumeric edited or numeric edited items.

COUNT IN Phrase: The data-count fields for each data transfer are identifier-6, identifier-9, and so on. These identifiers must be described as elementary numeric integer data items; they cannot contain the symbol P in their PICTURE clauses. Each field holds the count of delimited characters in the sending field to be transferred to this receiving field; the delimiters are not included in this count.

POINTER Phrase: The pointer field is identifier-10. This identifier must be described as an elementary numeric integer data item; it cannot contain the symbol P in its PICTURE clause. The identifier contains a value position in the sending field. When this phrase is specified, this field must be initialized before processing of the UNSTRING statement to a value that is not less than one and not greater than the count of the sending field.

TALLYING Phrase: The field-count is identifier-11. This identifier must be described as an elementary numeric integer data item; it cannot contain the symbol P in its PICTURE clause. The identifier is incremented by the number of data receiving fields acted upon in this processing of the UNSTRING statement. When this phrase is specified, this field must be initialized before processing of the UNSTRING statement.

The data-count fields, the pointer field, and the field-count field must each be integer items without the symbol P in the PICTURE character-strings.

UNSTRING Statement Processing

When the UNSTRING statement is initiated, the current data receiving field is identifier-4. Data is transferred from the sending field to the current data receiving field according to the following rules:

- If the POINTER phrase is not specified, the sending field character-string is examined beginning with the leftmost character. If the POINTER phrase is specified, the field is examined beginning at the relative character position specified by the value in the pointer field.
- If the DELIMITED BY phrase is specified, the examination proceeds left to right character by character until a delimiter is encountered. If the end of the sending field is reached before a delimiter is found, the examination ends with the last character in the sending field.
- If the DELIMITED BY phrase is not specified, the number of characters examined is equal to the size of the current data receiving field, which depends on its data category:
  - If the receiving field is alphanumeric or alphabetic, the number of characters examined is equal to the number of characters in the current receiving field.
  - If the receiving field is numeric, the number of characters examined is equal to the number of characters in the integer portion of the current receiving field.
  - If the receiving field is described with the SIGN IS SEPARATE clause, the characters examined are one fewer than the size of the current receiving field.
  - If the receiving field is described as a variable-length data item, the number of characters examined is determined by the current size of the current receiving field.
- The examined characters (excluding any delimiter characters) are treated as an alphanumeric elementary item, and are moved into the current data receiving field according to the rules for the MOVE statement.
- If the DELIMITER IN phrase is specified, the delimiting characters in the sending field are treated as an elementary alphanumeric item and are moved to the current delimiter receiving field according to the rules for the MOVE statement. If the delimiting condition is the end of the sending field, the current delimiter receiving field is filled with spaces.
- If the COUNT IN phrase is specified, a value equal to the number of examined characters (excluding any delimiters) is moved into the data count field, according to the rules for an elementary move.
- If the DELIMITED BY phrase is specified, the sending field is further examined, beginning with the first character to the right of the delimiter.
- If the DELIMITED BY phrase is not specified, the sending field is further examined, beginning with the first character to the right of the last character examined.
- After data is transferred to the first data receiving field (identifier-4), the current data receiving field becomes identifier-7. For each succeeding

current data receiving field, the preceding procedure is repeated–either until all of the characters in the sending field have been transferred, or until there are no more unfilled data receiving fields.

- When the POINTER phrase is specified, the contents of the pointer field behaves as if incremented by one for each examined character in the sending field. When this processing of the UNSTRING statement is completed, the pointer field contains a value equal to its initial value plus the number of characters examined in the sending field.
- When the TALLYING phrase is specified and the processing of the UNSTRING statement is completed, the tallying identifier contains a value equal to the initial value plus the number of data receiving areas acted upon; this count includes any null fields.
- When an overflow condition exists, the processing of the UNSTRING statement is terminated. If the ON OVERFLOW phrase has been specified, that imperative-statement is processed. If the ON OVERFLOW phrase has not been specified, control passes to the next executable statement. An overflow condition exists when:
  - An UNSTRING statement is initiated and the value in the pointer field is less than 1 or greater than the length of the sending field.
  - Or, all data receiving fields have been acted upon during UNSTRING statement processing, and the sending field still contains unexamined characters.

Note: If any of the UNSTRING statement identifiers are subscripted or indexed, the subscripts and indexes are evaluated as follows:

- Any subscripting or indexing associated with the sending field, the pointer field, or the field-count field is evaluated only once–immediately before any data is transferred.
- Any subscripting or indexing associated with the delimiters, the data and delimiter receiving fields or the data-count fields, is evaluated immediately before the transfer of data into the affected data item.

Figure 85 on page 449 illustrates the processing rules for the UNSTRING statement.

```
The following UNSTRING statement has the processing results shown:

UNSTRING ID-SEND DELIMITED BY DEL-ID OR ALL "*"
    INTO ID-R1 DELIMITER IN ID-D1 COUNT IN ID-C1     (All the data
         ID-R2 DELIMITER IN ID-D2                    receiving fields
         ID-R3 DELIMITER IN ID-D3 COUNT IN ID-C3     are defined as
         ID-R4 COUNT IN ID-C4                        alphanumeric.)
    WITH POINTER ID-P
    TALLYING IN ID-T
    ON OVERFLOW GO TO OFLOW-EXIT.
```

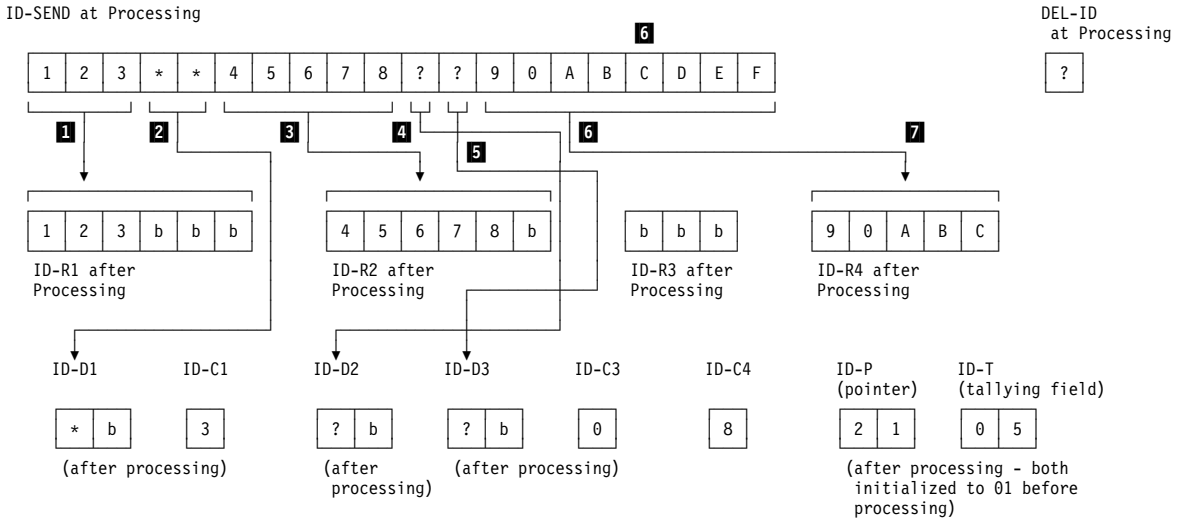


Figure 85. UNSTRING Statement Processing Results

The order of processing is:

1 Three characters are examined before a delimiter is encountered. These characters are moved to ID-R1.

2 The delimiter "*" is placed in ID-D1; the number of characters before a delimiter is moved to ID-C1.

Note: Because ALL "*" is specified, the second asterisk is treated as part of the delimiter. Only the first occurrence of the delimiter is moved to the DELIMITER IN data item, ID-D1.

3 Five characters are examined before the next delimiter is encountered. These characters are moved to ID-R2.

4 The delimiter "?" is placed in ID-D2.

Note: Because ALL is not specified for this delimiter, only the first "?" is considered to be the delimiter. The second "?" is the delimiter for the next receiving field.

5 No characters are examined before the next delimiter is encountered, so no characters are moved to ID-R3. Padding causes ID-R3 to be filled with spaces. The delimiter "?" is placed in ID-D3. The number of examined characters (0) is moved to ID-C3.

6 The remaining 8 characters are examined; no delimiter is found. These characters are moved to ID-R4. Because of the size of the receiving field, the last 3 characters are lost due to truncation. However, the total number of characters examined (8) is moved to ID-C4.

7 After processing, ID-P is incremented by the number of characters examined, and ID-T is incremented by the number of receiving fields processed.

UNSTRING Statement Example

The following example illustrates some of the considerations that apply to the UNSTRING statement.

In the Data Division, the user has defined the following input record to be acted upon by the UNSTRING statement:

```
01 INV-RCD.
   05 CONTROL-CHARS  PIC XX.
   05 ITEM-INDENT    PIC X(20).
   05 FILLER         PIC X.
   05 INV-CODE       PIC X(10).
   05 FILLER         PIC X.
   05 NO-UNITS       PIC 9(6).
   05 FILLER         PIC X.
   05 PRICE-PER-M    PIC 99999.
   05 FILLER         PIC X.
   05 RTL-AMT        PIC 9(6).99.
```

The next two records are defined as receiving fields for the UNSTRING statement. DISPLAY-REC is to be used for printed output. WORK-REC is to be used for further internal processing.

```
01 DISPLAY-REC
   05 INV-NO        PIC X(6).
   05 FILLER        PIC X VALUE SPACE
   05 ITEM-NAME     PIC X(20).
   05 FILLER        PIC X VALUE SPACE
   05 DISPLAY-DOLS  PIC 9(6).

01 WORK-REC
   05 M-UNITS       PIC 9(6).
   05 FIELD-A       PIC 9(6).
   05 WK-PRICE
      REDEFINES
      FIELD-A       PIC 9999V99.
   05 INV-CLASS     PIC X(3).
```

The user has also defined the following fields for use as control fields in the UNSTRING statement.

```
01 DBY-1          PIC X, VALUE IS ".".
01 CTR-1          PIC 99, VALUE IS ZERO.
01 CTR-2          PIC 99, VALUE IS ZERO.
01 CTR-3          PIC 99, VALUE IS ZERO.
01 CTR-4          PIC 99, VALUE IS ZERO.
01 DLTR-1         PIC X.
01 DLTR-2         PIC X.
01 CHAR-CT        PIC 99, VALUE IS 3.
01 FLDS-FILLED    PIC 99, VALUE IS ZERO.
```

In the Procedure Division, the user writes the following UNSTRING statement to move subfields of INV-RCD to the subfields of DISPLAY-REC and WORK-REC:

```
UNSTRING INV-RCD
 DELIMITED BY ALL SPACES
   OR "/"
   OR DBY-1
 INTO ITEM-NAME COUNT IN CTR-1,
 INV-NO DELIMITER IN DLTR-1
   COUNT IN CTR-2,
 INV-CLASS,
 M-UNITS COUNT IN CTR-3,
 FIELD-A,
 DISPLAY-DOLS DELIMITER IN DLTR-2
   COUNT IN CTR-4
 WITH POINTER CHAR-CT
 TALLYING IN FLDS-FILLED
 ON OVERFLOW
   GO TO UNSTRING-COMPLETE.
```

Before the UNSTRING statement is issued, the user places the value 3 in the CHAR-CT (the pointer item), so as not to work with the two control characters at the beginning of INV-RCD. In DBY-1, a period is placed for use as a delimiter, and in FLDS-FILLED (the tallying item) the value 0 is placed. The following data is then read into INV-RCD as shown in Figure 86.

```
Column

1         10        20          30          40          50          60
|         |         |           |           |           |           |
ZYFOUR-PENNY-NAILS      707890/BBA   475120  00122  000379.50
```

Figure 86. UNSTRING Statement Example–Input Data

When the UNSTRING statement is processed, the following actions take place:

1. Positions 3 through 18 (FOUR-PENNY-NAILS) of INV-RCD are placed in ITEM-NAME, left-justified within the area, and the unused character positions are padded with spaces. The value 16 is placed in CTR-1.
2. Because ALL SPACES is specified as a delimiter, the five contiguous SPACE characters are considered to be one occurrence of the delimiter.
3. Positions 24 through 29 (707890) are placed in INV-NO. The delimiter character / is placed in DLTR-1, and the value 6 is placed in CTR-2.
4. Positions 31 through 33 are placed in INV-CLASS. The delimiter is a SPACE, but because no field has been defined as a receiving area for delimiters, the SPACE is merely bypassed.
5. Positions 35 through 40 (475120) are examined and are placed in M-UNITS. The delimiter is a SPACE, but because no receiving field has been defined as a receiving area for delimiters, the SPACE is bypassed. The value 6 is placed in CTR-3.
6. Positions 42 through 46 (00122) are placed in FIELD-A and right-justified within the area. The high-order digit position is filled with a 0 (zero). The delimiter is

a SPACE, but because no field has been defined as a receiving area for delimiters, the SPACE is bypassed.

7. Positions 48 through 53 (000379) are placed in DISPLAY-DOLS. The period delimiter character is placed in DLTR-2, and the value 6 is placed in CTR-4.
8. Because all receiving fields have been acted upon and two characters of data in INV-RCD have not been examined, the ON OVERFLOW exit is taken, and processing of the UNSTRING statement is completed.

At the end of processing of the UNSTRING statement, DISPLAY-REC contains the following data:

```
707890 FOUR-PENNY-NAILS     000379
```

WORK-REC contains the following data:

```
475120000122BBA
```

CHAR-CT (the pointer field) contains the value 55, and FLD-FILLED (the tallying field) contains the value 6.

Note: One UNSTRING statement can be written instead of a series of MOVE statements.

Procedure Branching Statements

Statements, sentences, and paragraphs in the Procedure Division are ordinarily processed sequentially. The procedure branching statements allow alterations in the sequence. The procedure branching statements are: ALTER, EXIT, GO TO, PERFORM, and STOP.

ALTER Statement

The ALTER statement changes the transfer point specified in a GO TO statement.

Format

```
ALTER procedure-name-1 TO [ PROCEED TO ] procedure-name-2

    [ , procedure-name-3 TO [ PROCEED TO ] procedure-name-4 ]  . . .
```

Procedure-name-1, procedure-name-3, and so on, must each name a Procedure Division paragraph that contains only one sentence. That sentence must be a GO TO statement without the DEPENDING ON phrase.

Procedure-name-2, procedure-name-4, and so on, must each name a Procedure Division section or paragraph.

ALTER statement processing modifies the GO TO statement in the paragraph named by procedure-name-1, procedure-name-3, and so on. Subsequent processing of the modified GO TO statement(s) cause control to be transferred to procedure-name-2, and (if specified) procedure-name-4, and so on. For example:

```
PARAGRAPH-1.
  GO TO BYPASS-PARAGRAPH.
PARAGRAPH-1A.
  .
  .
BYPASS-PARAGRAPH.
  .
  .
 ALTER PARAGRAPH-1 TO PROCEED TO
     PARAGRAPH-2.
  .
  .
PARAGRAPH-2.
  .
  .
```

Before the ALTER statement is processed, when control reaches PARAGRAPH-1, the GO TO statement transfers control to BYPASS-PARAGRAPH. After processing of the ALTER statement, however, the next time control reaches PARAGRAPH-1, the GO TO statement transfers control to PARAGRAPH-2.

Note: The ALTER statement acts as a program switch, allowing, for example, one sequence of processing during initialization and another sequence during the bulk of file processing. Because altered GO TO statements are difficult to debug, it is preferable to test a switch, and based on the value of the switch, process a particular code sequence.

Segmentation Information

A GO TO statement in a section whose segment-number is greater than or equal to 50 must not be referred to by an ALTER statement in a section with a different segment-number. All other uses of the ALTER statement are valid and are processed.

Modified GO TO statements in independent segments can sometimes be returned to their initial states. See “Program Segments” on page 503 and “Independent Segments” on page 503 for further discussion.

EXIT Statement

The EXIT statement provides a common end point for a series of procedures.

Format

```
EXIT [ PROGRAM ].
```

The EXIT statement must appear in a sentence by itself, and this sentence must be the only sentence in the paragraph. The EXIT statement lets the user assign a procedure-name to a given point in a program.

The EXIT statement has no other effect on the compilation or running of the program.

The EXIT PROGRAM statement is discussed under “Procedure Division–Inter-Program Communication” on page 509.

Note: The EXIT statement is useful for documenting the end point in a series of procedures. If an exit paragraph is written as the last paragraph in a Declarative procedure or a series of performed procedures, it identifies the point at which control will be transferred. When control reaches such an exit paragraph and the associated Declarative or PERFORM statement is active, control is transferred to the appropriate part of the Procedure Division. When control reaches such an exit paragraph and no associated PERFORM statement or Declarative procedure is active, control passes through the EXIT statement to the first statement of the next paragraph.

If an EXIT statement is not written, the end of the sequence is difficult to determine unless the user knows the logic of the program.

GO TO Statement

The GO TO statement transfers control from one part of the Procedure Division to another. The formats of the GO TO statement are as follows:

Format 1

```
GO TO [ procedure-name-1 ]
```

Format 2

```
GO TO procedure-name-1 [ , procedure-name-2 ] . . . , procedure-name-n

   DEPENDING ON identifier
```

Each procedure-name specified must name a paragraph or section in the Procedure Division.

Format 1–Unconditional GO TO

The GO TO statement transfers control to the first statement in the paragraph or section named in procedure-name-1 unless the GO TO statement has been modified by an ALTER statement.

When a Format 1 GO TO statement appears in a sequence of imperative statements, it must be the last statement in the sequence.

When a paragraph is referred to by an ALTER statement, the paragraph can consist only of a paragraph-name followed by a Format 1 GO TO statement.

If procedure-name-1 is not specified in a Format 1 GO TO statement, an ALTER statement must have been processed before the processing of the GO TO statement. The GO TO statement must immediately follow a paragraph-name and must be the only statement in the paragraph.

Format 2–Conditional GO TO

Control is transferred to one of a series of procedures, depending on the value of identifier. Identifier must name an elementary integer item. When identifier has a value of one, control is transferred to the first statement in the procedure named by procedure-name-1; if it has a value of two, control is transferred to the first statement in the procedure named by procedure-name-2, and so on.

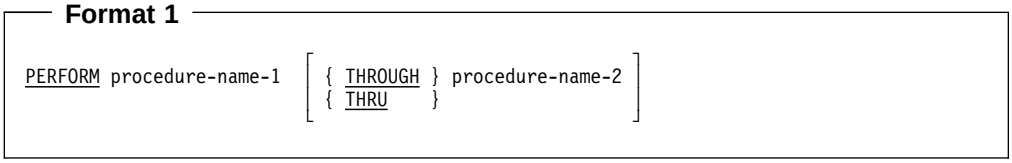
If the value of identifier is anything other than a value within the range 1 through n (where n is the number of procedure-names specified in this GO TO statement), the GO TO statement is ignored. Instead, control passes to the next statement in the normal sequence of processing.

The maximum number of procedure-names permitted for a Format 2 GO TO statement is 255.

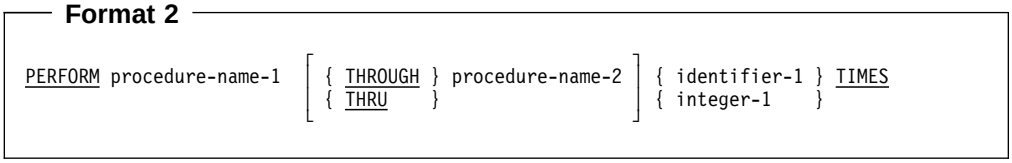
PERFORM Statement

The PERFORM statement transfers control explicitly to one or more procedures and implicitly returns control to the next executable statement after processing of the specified procedure(s) is completed. The formats of the PERFORM statement are as follows:

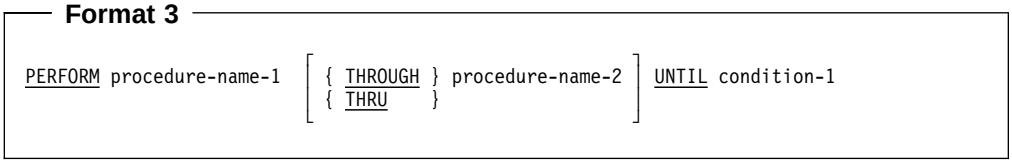
Format 1



Format 2



Format 3



Format 4

```
PERFORM procedure-name-1  [ { THROUGH } procedure-name-2 ]
                          [ { THRU    }                  ]

   VARYING { identifier-1 } FROM { identifier-2 }
           { index-name-1 }      { index-name-2 }
                                 { literal-2    }

   BY { identifier-3 } UNTIL condition-1
      { literal-3    }

   [ AFTER { identifier-4 } FROM { identifier-5 } ]
   [       { index-name-4 }      { index-name-5 } ]
   [                             { literal-5    } ]

   BY { identifier-6 } UNTIL condition-2
      { literal-6    }

   [ AFTER { identifier-7 } FROM { identifier-8 } ]
   [       { index-name-7 }      { index-name-8 } ]
   [                             { literal-8    } ]

   BY { identifier-9 } UNTIL condition-3 ] ]
      { literal-9    }                  | |
                                        | |
                                        ] ]
```

Each procedure-name must name a section or paragraph in the Procedure Division.

When both procedure-name-1 and procedure-name-2 are specified, if either is a procedure-name in a Declarative procedure, then both must be procedure-names in the same Declarative procedure.

Each identifier must name a numeric elementary item.

Each literal must be a numeric literal.

The set of statements within the range of procedure-name-1 (through procedure-name-2 if specified) for a PERFORM statement are referred to as the *specified set of statements.*

Whenever a PERFORM statement is processed, control is transferred to the first statement of the specified set of statements. Control is always returned to the statement following the PERFORM statement. The point from which this control is returned is determined as follows:

- If procedure-name-1 is a paragraph name and procedure-name-2 is not specified, the return is made after the processing of the last statement of procedure-name-1.
- If procedure-name-1 is a section-name and procedure-name-2 is not specified, the return is made after the processing of the last sentence of the last paragraph in that section.
- If procedure-name-2 is specified and it is a paragraph name, the return is made after the processing of the last statement of that paragraph.

- If procedure-name-2 is specified and it is a section-name, the return is made after the processing of the last sentence of the last paragraph in the section.

The only necessary relationship between procedure-name-1 and procedure-name-2 is that a consecutive sequence of operations is processed beginning at the procedure named by procedure-name-1 and ending with the processing of the procedure named by procedure-name-2.

When both procedure-name-1 and procedure-name-2 are specified, GO TO and PERFORM statements can appear within the sequence of statements contained in these paragraphs or sections. A GO TO statement should not refer to a procedure-name outside the range of procedure-name-1 through procedure-name-2. If this is done, results are unpredictable and are not diagnosed.

When only procedure-name-1 is specified, PERFORM and GO TO statements can appear within the procedure. A GO TO statement should not refer to a procedure-name outside the range of procedure-name-1. If this is done, results are unpredictable and are not diagnosed.

When the performed procedures include another PERFORM statement, the sequence of procedures associated with the embedded PERFORM statement must be totally included in or totally excluded from the performed procedures of the first PERFORM statement. That is, an active PERFORM statement whose processing point begins within the range of performed procedures of another active PERFORM statement must not allow control to pass through the exit point of the other active PERFORM statement. In addition, two or more such active PERFORM statements must not have a common exit.

IBM Extension

Two active PERFORM statements can have a common exit point.

End of IBM Extension

When control passes to the sequence of procedures by means other than a PERFORM statement, control passes through the exit point to the next executable statement as if no PERFORM statement referred to these procedures.

The range of a PERFORM statement logically consists of all processed statements resulting from the processing of a PERFORM statement, and includes the implicit transfer of control to the end of the PERFORM statement. This range includes all processed statements in Declarative procedures as well as statements resulting from the transfer of control by CALL, EXIT without the PROGRAM phrase, GO TO, and PERFORM statements. The statements in the range of a PERFORM statement need not appear consecutively in the source program.

Figure 87 on page 458 illustrates valid sequences of processing for PERFORM statements.

The preceding rules refer to all four formats of the PERFORM statement. The following sections give rules applying to each individual format.

Format 1

Format 1 is the basic PERFORM statement. The procedure(s) referred to is processed once, and then control passes to the next executable statement following the PERFORM statement.

Format 2

Format 2 uses the TIMES phrase. Identifier-1 must name an integer item. The procedure(s) referred to is processed the number of times specified by the value in identifier-1 or integer-1. Control then passes to the next executable statement following the PERFORM statement. The following rules apply:

- If identifier-1 is zero or a negative number at the time the PERFORM statement is initiated, control passes to the statement following the PERFORM statement.
- After the PERFORM statement has been initiated, any reference to identifier-1 or change in the value of identifier-1 has no effect in varying the number of times the procedures are run.

Format 3

Format 3 uses the UNTIL phrase. The procedure(s) referred to is processed until the condition specified by the UNTIL phrase is true. Control is then passed to the next executable statement following the PERFORM statement.

If condition-1 is true at the time the PERFORM statement is encountered, the specified procedure(s) is not processed.

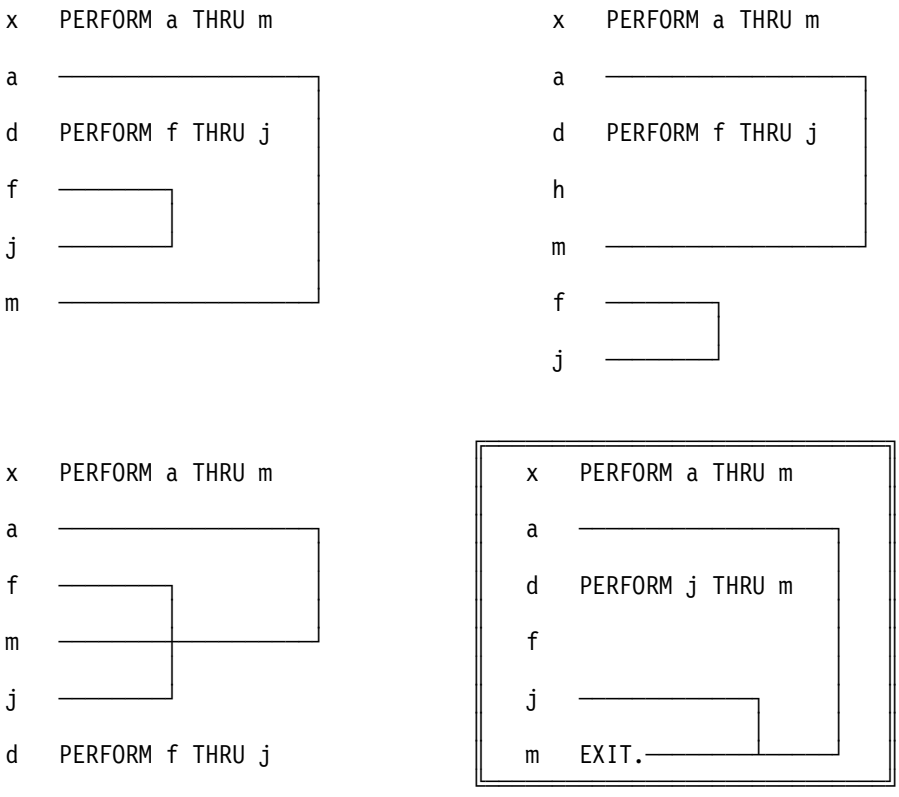


Figure 87. Valid PERFORM Statement Processing Sequences

Format 4

Format 4 uses the VARYING phrase. This phrase increments or decrements one or more identifiers or index-names according to the following rules. Once the condition(s) specified in the UNTIL phrase is satisfied, control is passed to the next executable statement following the PERFORM statement.

No matter how many variables are specified, the following rules apply:

- In the VARYING/AFTER phrases, when an index-name is specified:
  - The index-name is initialized and incremented or decremented according to the rules for the SET statement. For a description of the SET statement, see "TABLE HANDLING" on page 469.
  - In the associated FROM phrase, an identifier must be described as an integer and have a positive value; a literal must be a positive integer.
  - In the associated BY phrase, an identifier must be described as an integer; a literal must be a nonzero integer.
- In the FROM phrase, when an index-name is specified:
  - In the associated VARYING/AFTER phrase, an identifier must be described as an integer. It is initialized as described in the SET statement.
  - In the associated BY phrase, an identifier must be described as an integer and have a nonzero value; a literal must be a nonzero integer.
- In the BY phrase, identifiers and literals must have a nonzero value.
- Changing the values of identifiers and/or index-names in the VARYING, AFTER, FROM, and BY phrases during processing changes the number of times the procedures are run.

The way in which operands are incremented or decremented depends on the number of variables specified.

Varying One Identifier

In the following discussion, every reference to identifier-n refers equally to index-name-n except when identifier-n is the object of the BY phrase.

The following actions take place:

1. Identifier-1 is set equal to its starting value, identifier-2 or literal-2.
2. Condition-1 is evaluated:
   a. If it is false, steps 3 through 5 are processed.
   b. If it is true, control passes directly to the statement following the PERFORM statement.
3. Procedure-1 through procedure-2 (if specified) are processed once.
4. Identifier-1 is augmented by identifier-3 (or literal-3), and condition-1 is evaluated again.
5. Steps 2 through 4 are repeated until condition-1 is true.

Figure 88 on page 460 is a flowchart illustrating the logic of the PERFORM statement when one identifier is varied.

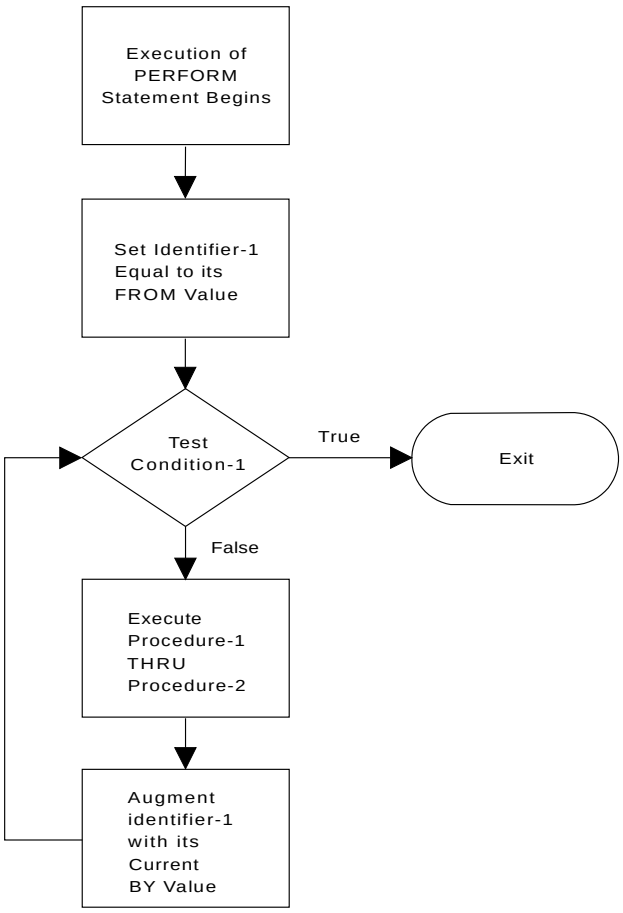


Figure 88. Format 4–PERFORM Statement Logic–Varying One Identifier

The following example shows a `PERFORM` statement varying one identifier. This `PERFORM` logic is processed 100 times.

```
.. 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... 7

   DATA DIVISION.
   WORKING-STORAGE SECTION.
   01  SUB1                   PIC 999.
   01  TOTAL-HOLD             PIC 99      VALUE 57.
   01  HOLD-2                 PIC 99      VALUE 10.
   01  HOLD-THE-SUM           PIC 99      VALUE ZERO.
   01  TABLE-ELEMENT.
       03  ELEMENTS-OF-TABLE  PIC 9       OCCURS 100 TIMES.
   PROCEDURE DIVISION.
   100-START-PROCESSING.
  *    THIS PERFORM LOGIC IS PROCESSED 100 TIMES.
       PERFORM SAMPLE-PERFORM THRU PERFORM-EXIT
           VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 GREATER THAN 100.
  *    THIS ADD STATEMENT IS PROCESSED AFTER PERFORM IS DONE.
       ADD TOTAL-HOLD HOLD-2 GIVING HOLD-THE-SUM.
       DISPLAY "TOTAL OF TWO VARIABLES = " HOLD-THE-SUM.
       PERFORM ANOTHER-WAY-TO-INITIALIZE THRU AWTI-EXIT.
  *    THE TABLE WILL BE ALL ZEROS AND SHOULD PRINT AS SUCH.
       DISPLAY "THE TABLE  " TABLE-ELEMENT.
       STOP RUN.
    SAMPLE-PERFORM.
       MOVE ZEROS TO ELEMENTS-OF-TABLE (SUB1).
    PERFORM-EXIT.
       EXIT.
    ANOTHER-WAY-TO-INITIALIZE.
       MOVE ZEROS TO TABLE-ELEMENT.
    AWTI-EXIT.
       EXIT.
```

Varying Two Identifiers

In the following discussion, every reference to identifier-n refers equally to index-name-n except when identifier-n is the object of the BY phrase.

The following actions take place:

1. Identifier-1 and identifier-4 are set to their initial values, identifier-2 (or literal-2) and identifier-5 (or literal-5), respectively.
2. Condition-1 is evaluated:
   a. If it is false, steps 3 through 7 are processed.
   b. If it is true, control passes directly to the statement following the PERFORM statement.
3. Condition-2 is evaluated:
   a. If it is false, steps 4 through 6 are processed.
   b. If it is true, identifier-4 is set to the current value of identifier-5, and identifier-1 is augmented by identifier-3 (or literal-3), and step 2 is repeated.
4. Procedure-1 through procedure-2 (if specified) are run once.
5. Identifier-4 is augmented by identifier-6 (or literal-6).
6. Steps 3 through 5 are repeated until condition-2 is true.
7. Steps 2 through 6 are repeated until condition-1 is true.

At the end of PERFORM statement processing, identifier-4 contains the current value of identifier-5. Identifier-1 has a value that exceeds the last used setting by the increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement processing, in which case identifier-1 contains the current value of identifier-2).

Figure 89 on page 463 is a flowchart illustrating the logic of the PERFORM statement when two identifiers are varied.

The following example shows a PERFORM statement varying two identifiers. This PERFORM logic is processed 126 times. This program searches a table and gives a total of female employees.

```
.. 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7

  DATA DIVISION.
  FILE SECTION.
  FD  PRINTED-REPORT
      LABEL RECORDS OMITTED.
  01  PRINT-OUT             PIC X(132).
  FD  EMPLOYEE-DATA
      BLOCK CONTAINS 1 RECORDS
      RECORD CONTAINS 80 CHARACTERS
      LABEL RECORDS STANDARD
      DATA RECORD IS EMPLOYEE-RECORD.
  01  EMPLOYEE-RECORD        PIC X(90).
  WORKING-STORAGE SECTION.
  01  RECORDS-IN             PIC 9(5)    VALUE ZEROS.
  01  EOF-SW                 PIC X       VALUE "N".
  01  HOLD-INPUT-RECORD.
      03  EMPLOYEE-SEX       PIC 9.
          88 MALE            VALUE IS 1.
          88 FEMALE          VALUE IS 2.
      03  EMPLOYEE-RACE      PIC 9.
          88 RACE-CODES      VALUES ARE 1 THRU 7.
      03  EMPLOYEE-JOB-CLASS PIC 99.
          88 JOB-CLASS       VALUES ARE 01 THRU 18.
      03  FILLER             PIC X(76)   VALUE SPACES.
  01  EMPLOYEE-TABLE.
      03  E-SEX              OCCURS  2 TIMES.
          05  E-RACE           OCCURS  7 TIMES.
              07 E-JOB      OCCURS 18 TIMES PIC 99.
  01  SUB1                   PIC 99.
  01  SUB2                   PIC 99.
  01  SUB3                   PIC 99.
  01  TOTAL-WOMEN            PIC 9(5)    VALUE ZEROS.
  PROCEDURE DIVISION.
  100-START-IT.
      OPEN INPUT EMPLOYEE-DATA OUTPUT PRINTED-REPORT.
      MOVE ZEROS TO EMPLOYEE-TABLE.
  200-READ-IT.
      READ EMPLOYEE-DATA RECORD INTO HOLD-INPUT-RECORD
          AT END MOVE "Y" TO EOF-SW.
      ADD 1 TO RECORDS-IN.
  300-MAINLINE-LOGIC.
 *    THE PERFORM STATEMENT USING 2 VARIABLES WILL BE DONE 126
 *    TIMES
      PERFORM LOAD-TABLE UNTIL EOF-SW = "Y".
```

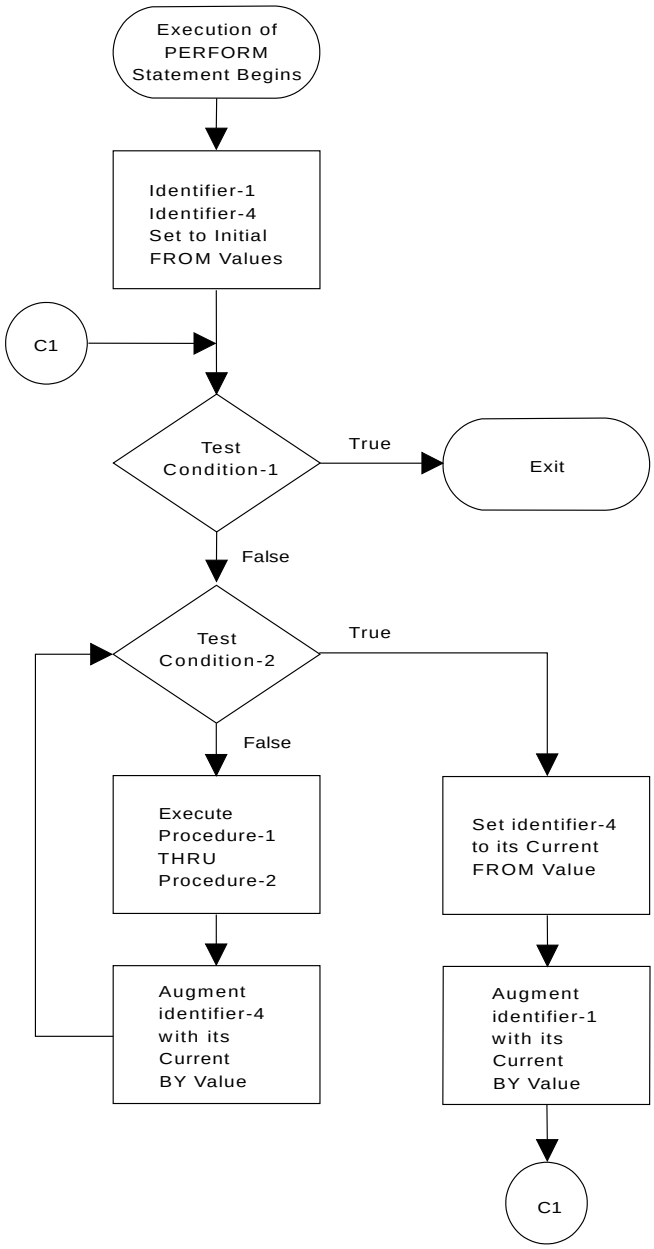


Figure 89. Format 4–PERFORM Statement Logic–Varying Two Identifiers

```
    PERFORM FIND-NUMBER-OF-WOMEN
        VARYING SUB2 FROM 1 BY 1 UNTIL SUB2 > 7
        AFTER SUB3 FROM 1 BY 1 UNTIL SUB3 > 18.
    PERFORM WRITE-REPORT THRU WR-EXIT.
    DISPLAY "TOTAL RECORDS IN " RECORDS-IN.
    STOP RUN.
LOAD-TABLE.
    MOVE EMPLOYEE-SEX TO SUB1.
    MOVE EMPLOYEE-RACE TO SUB2.
    MOVE EMPLOYEE-JOB-CLASS TO SUB3.
    ADD 1 TO E-JOB (SUB1 SUB2 SUB3).
    PERFORM 200-READ-IT.
FIND-NUMBER-OF-WOMEN.
    ADD E-JOB (2 SUB2 SUB3) TO TOTAL-WOMEN.
WRITE-REPORT.
```

```
    MOVE TOTAL-WOMEN TO PRINT-OUT.
    WRITE PRINT-OUT.
WR-EXIT.
    EXIT.
```

Varying Three Identifiers

In the following discussion, every reference to identifier-n refers equally to index-name-n except when identifier-n is the object of the BY phrase.

The actions are the same as for varying two identifiers except that identifier-7 goes through the complete cycle each time that identifier-4 is augmented by identifier-6 or literal-6, which in turn goes through a complete cycle each time identifier-1 is varied.

At the end of PERFORM statement processing, identifier-4 and identifier-7 contain the current values of identifier-5 and identifier-8, respectively. Identifier-1 has a value exceeding its last used setting by one increment/decrement value (unless condition-1 was true at the beginning of PERFORM statement processing, in which case identifier-1 contains the current value of identifier-2).

Figure 90 on page 465 is a flowchart illustrating the logic of the PERFORM statement when three identifiers are varied.

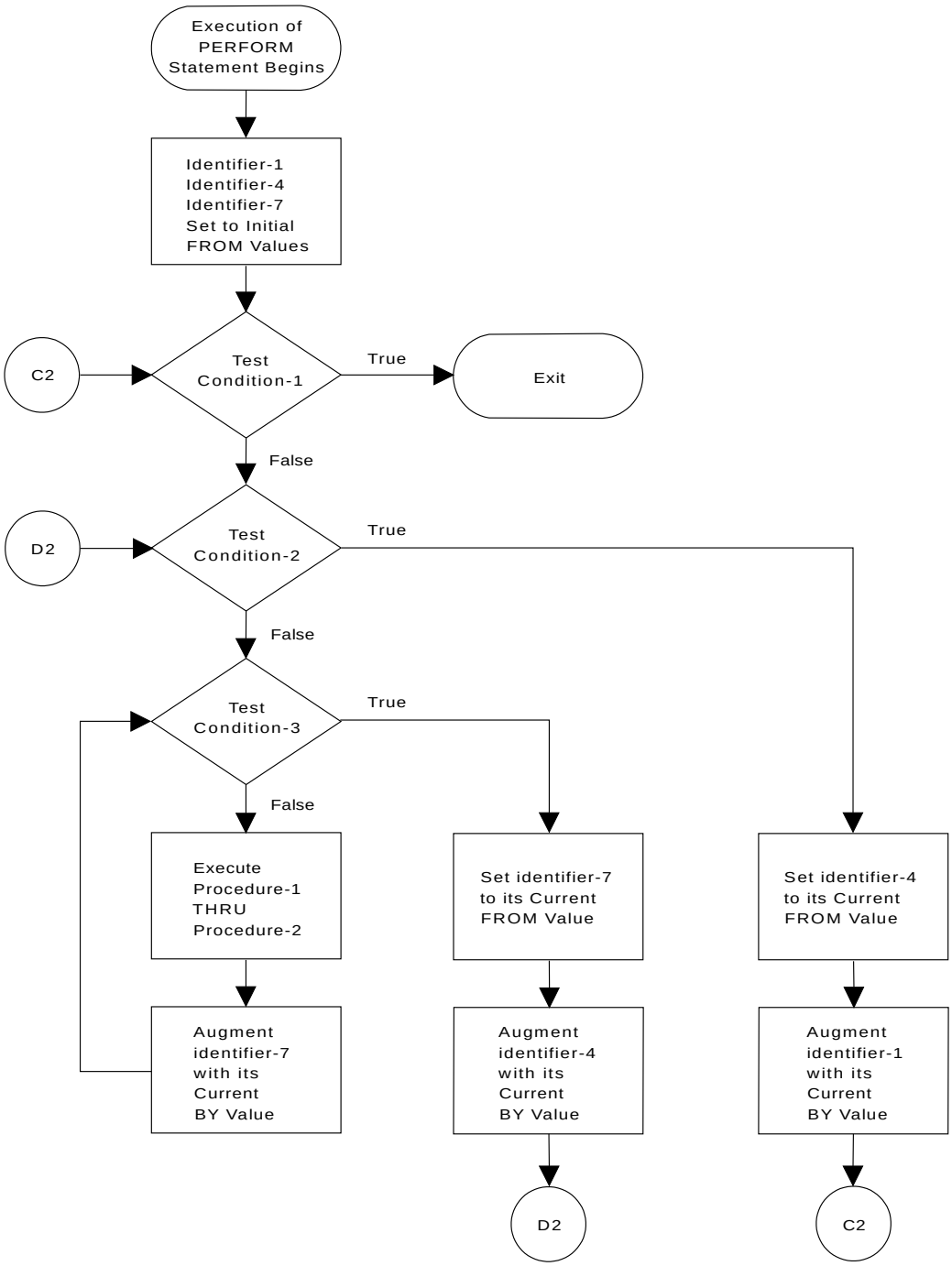


Figure 90. Format 4–PERFORM Statement Logic–Varying Three Identifiers

The following example shows a PERFORM statement varying three identifiers. This PERFORM logic is run 250 times.

```
.. 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  SUB1                   PIC 99.
 01  SUB2                   PIC 99.
 01  SUB3                   PIC 99.
 01  TEST-IT                PIC 99     VALUE 00.
 01  TOTAL-RECS             PIC 99     VALUE ZEROS.
 01  COMPANY-TABLE.
     05  DIVISION-IN        OCCURS 10 TIMES.
         10 DIVISION-NAME              PIC X(10).
         10 DIVISION-NUMBER            PIC 9(4).
         10 SECTION-IN      OCCURS 5 TIMES.
            15 UNIT-IN      OCCURS 5 TIMES.
               20 UNIT-NAME            PIC X(5).
               20 UNIT-NUMBER          PIC 9(4).
 PROCEDURE DIVISION.
 100-START-PROCESSING.
*    THIS PERFORM LOGIC IS PROCESSED 250 TIMES
     PERFORM ZERO-OUT-BIG-TABLE
         VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 10
*    SUB1 IS VARIED LAST
         AFTER   SUB2 FROM 1 BY 1 UNTIL SUB2 > 5
*    SUB2 IS VARIED SECOND
         AFTER   SUB3 FROM 1 BY 1 UNTIL SUB3 > 5.
*    SUB3 IS VARIED FIRST
     PERFORM ADDRESS-THE-VARIABLES THRU ATV-EXIT.
     DISPLAY "VARIABLE TEST-IT = " TEST-IT.
     STOP RUN.
 ZERO-OUT-BIG-TABLE.
     MOVE ZEROS TO UNIT-IN (SUB1 SUB2 SUB3).
 ADDRESS-THE-VARIABLES.
     IF UNIT-NUMBER OF UNIT-IN OF SECTION-IN OF DIVISION-IN
         OF COMPANY-TABLE (3 4 5) = 0
         ADD 1 TO TEST-IT.
 ATV-EXIT.
     EXIT.
```

Note: The procedures run by a PERFORM statement are, in effect, a closed subroutine that can be entered from many other points in the program.

The Format 4 PERFORM statement is especially useful in table handling. One Format 4 PERFORM statement can serially search an entire three-dimensional table.

Segmentation Information

A PERFORM statement appearing in a permanent segment can have in its range only one of the following:

- Sections, each of which has a segment number less than 50
- Sections and/or paragraphs wholly contained in a single independent segment.

A PERFORM statement that appears in an independent segment can have in its range only one of the following:

- Sections, each of which has a segment number less than 50

- Sections and/or paragraphs wholly contained within the same independent segment as the PERFORM statement.

Control is passed to the performed procedures only once for each processing of the PERFORM statement.

STOP Statement

The STOP statement halts the object program either temporarily or permanently.

Format

```
STOP { RUN     }
     { literal }
```

The literal can be numeric or nonnumeric, and can be any figurative constant except ALL literal. If the literal is numeric, it must be an unsigned integer.

When STOP literal is specified, the literal is communicated to the system operator for batch jobs and to the work station for interactive jobs. Program processing is suspended. Processing is resumed only after operator intervention.

The operator response determines whether the run unit continues at the next executable statement in the sequence, or a STOP RUN is processed.

Operator Response	Action
G (default)	Continue at next instruction.
C	Terminate processing of the run unit. Escape message CBE9001 is issued to the caller of the COBOL run unit. For batch jobs, the job is canceled if the CNLSEV parameter for the job contains a value that is less than or equal to the severity of the message.

The output of the STOP literal contains the program-name followed by the literal.

If the literal cannot be contained in the length of one line of the display device, the Help key must be used to display the entire literal.

When STOP RUN is specified, processing of the run unit is terminated. If a STOP RUN statement appears in a sequence of imperative statements, it must be the last or the only statement in the sequence. All files should be closed before a STOP RUN statement is processed. If you do not close the files, they are closed by compiler generated code. An implicit STOP RUN is always generated after the last statement in the source program.

Note: The STOP literal statement is useful for special situations when operator intervention is needed during program processing.

Compiler-Directing Statements

Compiler-directing statements provide instructions to the COBOL compiler. The compiler-directing statements are COPY, ENTER, and USE.

Only the ENTER statement is discussed in this chapter. In Chapter 7, "System/38-Compatible COBOL Programming Considerations," the COPY statement is discussed. The USE statement is discussed under "Declaratives" on page 364 in this chapter and under "DEBUGGING FEATURES" on page 517.

ENTER Statement

The COBOL/38 compiler does not allow modules written in another source language to be incorporated into COBOL programs. Therefore, the ENTER statement is not required or used by the COBOL/38 compiler, but is only treated as a comment. See the "CALL Statement" on page 510 for details about incorporating other object programs in COBOL programs.

Format

```
****************************************
* ENTER language-name [ routine-name ] . *
****************************************
```

Language-name and routine-name can be any user-defined word. The compiler expects a valid COBOL statement to immediately follow the ENTER statement.

Chapter 11. Using the Additional COBOL Functions

System/38-Compatible COBOL offers several additional functions that are useful to programmers who are writing more advanced applications. The additional functions provided by System/38-Compatible COBOL discussed in this chapter are:

- Table Handling
- SORT-MERGE
- Segmentation
- Inter-Program Communication
- Debugging
- FIPS Flagger.

TABLE HANDLING

Tables are often used in data processing. A table is a set of logically consecutive items, each of which has the same data description as the other items in the set. The items in a table can be described as separate contiguous items. However, this approach may not be satisfactory for two reasons. From a documentation stand-point, the homogeneity of the data items is not apparent; secondly, repetitive coding to reference unique data-names becomes a severe problem. Thus, a method of data reference is used which makes it possible to refer to all or to part of one table as an entity.

Table Handling Concepts

In System/38-Compatible COBOL, a table is defined with an OCCURS clause in its data description. The OCCURS clause specifies that the named item is to be repeated as many times as stated. The item so named is considered a table element, and its name and description apply to each repetition (or occurrence) of the item. Because the occurrences are not given unique data-names, reference to a particular occurrence can be made only by specifying the data-name of the table element, together with the occurrence number of the desired item within the element.

The occurrence number is known as a subscript and the technique of supplying the occurrence number of individual table elements is called subscripting. A related technique, called indexing, is also available for table references. Both subscripting and indexing are described in subsequent sections.

Table Definition

System/38-Compatible COBOL allows tables in one, two or three dimensions.

To define a one-dimensional table, the user writes an OCCURS clause as part of the definition of a table element. However, the OCCURS clause must not appear in a data description entry that has a 01, 66, 77, or 88 level-number. For example:

```
01 TABLE-ONE.
   05 ELEMENT-ONE OCCURS 3 TIMES.
      10 ELEMENT-A PIC X(4).
      10 ELEMENT-B PIC 9(4).
```

TABLE-ONE is the group item that contains the table. ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-ONE.

To define a two-dimensional table, a one-dimensional table is defined within each occurrence of another one-dimensional table. For example:

```
01 TABLE-TWO.
   05 ELEMENT-ONE OCCURS 3 TIMES.
      10 ELEMENT-TWO OCCURS 3 TIMES.
         15 ELEMENT-A PIC X(4).
         15 ELEMENT-B PIC 9(4).
```

TABLE-TWO is the group item that contains the table. ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE. ELEMENT-A and ELEMENT-B are elementary items subordinate to ELEMENT-TWO.

To define a three-dimensional table, a one-dimensional table is defined within each occurrence of another one-dimensional table, which is itself contained within each occurrence of another one-dimensional table. For example:

```
01 TABLE-THREE.
   05 ELEMENT-ONE OCCURS 3 TIMES.
      10 ELEMENT-TWO OCCURS 3 TIMES.
         15 ELEMENT-THREE OCCURS 2 TIMES
            PICTURE X(8).
```

TABLE-THREE is the group item that contains the table. ELEMENT-ONE is an element of a one-dimensional table that occurs three times. ELEMENT-TWO is an element of a two-dimensional table that occurs three times within each occurrence of ELEMENT-ONE. ELEMENT-THREE is an element of a three-dimensional table that occurs two times within each occurrence of ELEMENT-TWO. Figure 91 on page 471 shows the storage layout for TABLE-THREE.

ELEMENT-ONE Occurs Three Times	ELEMENT-TWO Occurs Three Times	ELEMENT-THREE Occurs Two Times	Byte Displacement
			0
ELEMENT-ONE (1)	ELEMENT-TWO (1, 1)	ELEMENT-THREE (1, 1, 1)	
			8
		ELEMENT-THREE (1, 1, 2)	
			16
	ELEMENT-TWO (1, 2)	ELEMENT-THREE (1, 2, 1)	
			24
		ELEMENT-THREE (1, 2, 2)	
			32
	ELEMENT-TWO (1, 3)	ELEMENT-THREE (1, 3, 1)	
			40
		ELEMENT-THREE (1, 3, 2)	
			48
ELEMENT-ONE (2)	ELEMENT-TWO (2, 1)	ELEMENT-THREE (2, 1, 1)	
			56
		ELEMENT-THREE (2, 1, 2)	
			64
	ELEMENT-TWO (2, 2)	ELEMENT-THREE (2, 2, 1)	
			72
		ELEMENT-THREE (2, 2, 2)	
			80
	ELEMENT-TWO (2, 3)	ELEMENT-THREE (2, 3, 1)	
			88
		ELEMENT-THREE (2, 3, 2)	
			96
ELEMENT-ONE (3)	ELEMENT-TWO (3, 1)	ELEMENT-THREE (3, 1, 1)	
			104
		ELEMENT-THREE (3, 1, 2)	
			112
	ELEMENT-TWO (3, 2)	ELEMENT-THREE (3, 2, 1)	
			120
		ELEMENT-THREE (3, 2, 2)	
			128
	ELEMENT-TWO (3, 3)	ELEMENT-THREE (3, 3, 1)	
			136
		ELEMENT-THREE (3, 3, 2)	
			144

Figure 91. Storage Layout for TABLE-THREE

Table References

Whenever the user refers to a table element, or to any item associated with a table element, the reference must indicate which occurrence is intended.

For a one-dimensional table, the occurrence number of the desired element gives the complete information. For tables of more than one dimension, an occurrence number for each dimension must be supplied. In the three-dimensional table defined in the previous discussion, for example, a reference to `ELEMENT-THREE` must supply the occurrence number for `ELEMENT-ONE`, `ELEMENT-TWO`, and `ELEMENT-THREE`. Either subscripting or indexing, described in the following paragraphs, can be used to supply the necessary references.

Subscripting

Subscripting is a method of providing table references through the use of subscripts. A subscript is an integer value that specifies the occurrence number of a table element. Subscripts can be used only when reference is made to an individual item within a table element.

Format

```
{ data-name-1    }  [ { OF } data-name-2 ]  . . . ( subscript-1 [ , subscript-2 [ , subscript-3 ] ] )
{ condition-name }  [ { IN }             ]
```

Data-name-1 must be the name of a table element. (Note that when qualification is used, it is data-name-1 that is subscripted, not data-name-2.)

The subscript can be represented either by a literal or a data-name.

A literal subscript must be an integer, and it must have a value of one or greater. The literal can have a positive sign or it may be unsigned. Negative subscript values are not permitted. For example, the following are valid literal subscript references to TABLE-THREE:

```
ELEMENT-THREE (1, 2, 1)

ELEMENT-THREE (2, 2, 1).
```

A data-name subscript must be described as an elementary numeric integer data item. A data-name subscript may be qualified; it may not be subscripted or indexed. For example, assuming that SUB1, SUB2, and SUB3 are all items subordinate to SUBSCRIPT-ITEM, valid data-name subscript references to TABLE-THREE are:

```
ELEMENT-THREE (SUB1, SUB2, SUB3)

ELEMENT-THREE IN TABLE-THREE (SUB1 OF
  SUBSCRIPT-ITEM, SUB2 OF SUBSCRIPT-ITEM,
  SUB3 OF SUBSCRIPT-ITEM)
```

The set of one to three subscripts must be written within a balanced pair of parentheses immediately following data-name-1 or its last qualifier. One or more spaces can optionally precede the opening parenthesis.

When more than one subscript is specified, each subscript must be separated from the next by either a space or a comma and a space.

When more than one subscript is required, the subscripts are written in the order of successively less inclusive data dimensions. For example, in the table reference ELEMENT-THREE (3, 2, 1), the first value (3) refers to the occurrence within ELEMENT-ONE, the second value (2) refers to the occurrence within ELEMENT-TWO, and the third value (1) refers to the occurrence within ELEMENT-THREE.

The lowest possible subscript value is 1; this value points to the first occurrence within the table element. The next sequential elements are pointed to by subscripts with values 2, 3, and so on. The highest permissible subscript value in any particular table element is the maximum number of occurrences specified in the OCCURS

clause. For example, in TABLE-THREE the highest possible subscript value for ELEMENT-ONE is 3, for ELEMENT-TWO is 3, and for ELEMENT-THREE is 2.

If the RANGE option is specified or implied (see "PROCESS Statement" on page 46), the system ensures that the subscript value is valid. If the RANGE option is not active, it is your responsibility to ensure that the subscript value is valid.

The RANGE option applies to subscripts only. The RANGE option does *not* verify that indexes are valid.

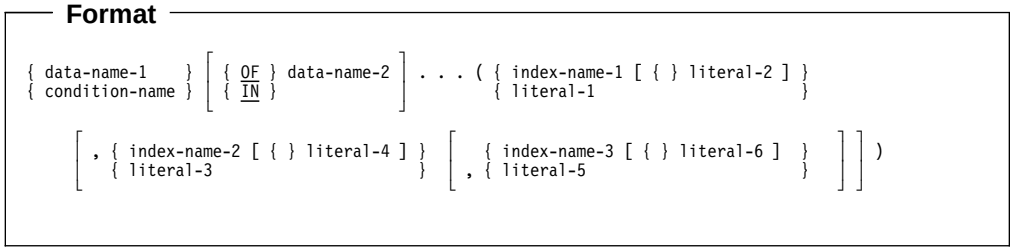
The following example shows subscripting using a three-level table. In this example, UNIT-NUMBER does not need qualification and could also be referenced as UNIT-NUMBER (3, 4, 5).

```
.. 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7

 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  SUB1                  PIC 99.
 01  SUB2                  PIC 99.
 01  SUB3                  PIC 99.
 01  TEST-IT               PIC 99    VALUE 00.
 01  TOTAL-RECS            PIC 99    VALUE ZEROS.
 01  COMPANY-TABLE.
     05  DIVISION-IN       OCCURS 10 TIMES.
         10 DIVISION-NAME             PIC X(10).
         10 DIVISION-NUMBER           PIC 9(4).
         10 SECTION-IN       OCCURS 5 TIMES.
            15 UNIT-IN       OCCURS 5 TIMES.
               20 UNIT-NAME           PIC X(5).
               20 UNIT-NUMBER         PIC 9(4).
 PROCEDURE DIVISION.
 100-START-PROCESSING.
     PERFORM ZERO-OUT-BIG-TABLE
         VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 10
*    SUB1 IS VARIED LAST
         AFTER   SUB2 FROM 1 BY 1 UNTIL SUB2 > 5
*    SUB2 IS VARIED SECOND
         AFTER   SUB3 FROM 1 BY 1 UNTIL SUB3 > 5.
*    SUB3 IS VARIED FIRST
     PERFORM ADDRESS-THE-VARIABLES THRU ATV-EXIT.
     DISPLAY "VARIABLE TEST-IT = " TEST-IT.
     STOP-RUN.
 ZERO-OUT-BIG-TABLE.
     MOVE ZEROS TO UNIT-IN (SUB1 SUB2 SUB3).
 ADDRESS-THE-VARIABLES.
     IF UNIT-NUMBER OF UNIT-IN OF SECTION-IN OF DIVISION-IN
         OF COMPANY-TABLE (3 4 5) = 0
         ADD 1 TO TEST-IT.
 ATV-EXIT.
     EXIT.
```

Indexing

Indexing is the method of providing table references through the use of indexes. An index is a compiler-generated storage area used to store table element occurrence numbers. For System/38-Compatible COBOL, the index contains a value that is an offset into the table.

Format



Data-name-1 must be the name of a table element. (Note that when qualification is used, it is data-name-1 that is indexed rather than data-name-2.)

Each index-name identifies an index to be used in table references. The index-name is specified through the INDEXED BY phrase in the OCCURS clause.

Each index named is a compiler-generated storage area, 2 bytes in length. Two forms of indexing are provided: direct and relative.

In direct indexing, the index-name is in the form of a subscript. In relative indexing, the index-name is followed by a space, a + or a -, another space, and an unsigned numeric literal. The literal is considered to be an occurrence number, and is converted to an index value before being added to or subtracted from the index-name index.

To be valid during processing, an index value must correspond to a table element occurrence number that is not less than one, or greater than the highest permissible occurrence number. This restriction applies to both direct and relative indexing.

The RANGE option (see “Create COBOL Program Command” on page 37) does *not* cause the system to verify that index values are valid. It is your responsibility to ensure valid index values.

An index-name must be initialized through a SET, PERFORM-Format 4, or SEARCH ALL statement before it is used in a table reference.

One or more index references (direct or relative) can be specified together with literal subscripts.

Further information on index-names is given later in this chapter in the description of the INDEXED BY phrase of the OCCURS clause.

Restrictions on Subscripting and Indexing

- A data-name must not be subscripted or indexed when it is being used as a subscript or qualifier.
- Indexing is not permitted when subscripting is not permitted.
- An index can be modified only by a PERFORM, SEARCH, or SET statement.
- When a literal is used in a subscript, it must be a positive or unsigned integer.
- When a literal is used in relative indexing, it must be an unsigned integer.

Table Initialization

A table can contain static values or dynamic values. Static values remain the same through every run of the object program. When this is true, the initial values of table elements can be specified in Working-Storage in one of two ways:

- The table can be described as a record containing contiguous subordinate data description entries, each of which contains a VALUE clause for the initial value. The record is then redescribed through a REDEFINES entry that contains a subordinate entry with an OCCURS clause. Because of the OCCURS clause, the subordinate entries of the redefined entry are repeated. For example:

```
01 TABLE-ONE.
   05 ELEMENT-ONE PICTURE X VALUE "1".
   05 ELEMENT-TWO PICTURE X VALUE "2".
   05 ELEMENT-THREE PICTURE X VALUE "3".
   05 ELEMENT-FOUR PICTURE X VALUE "4".
01 TABLE-TWO REDEFINES TABLE-ONE.
   05 OCCURS-ELEMENT OCCURS 4 TIMES
      PICTURE X.
```

- If the subordinate entries do not require separate handling, the VALUE of the entire entry can be given in the entry that names the table. The lower level entries then contain OCCURS clauses, and show the hierarchical structure of the table. The subordinate entries must not contain VALUE clauses. For example:

```
01 TABLE-ONE VALUE "1234".
   05 TABLE-TWO OCCURS 4 TIMES
      PICTURE X.
```

Dynamic values may change during one run of the object program, or from one run to another. If the dynamic values are always the same at the beginning of object program run, they can be initialized in the same manner as static values. If the initial values change from one run to the next, then the table can be defined without initial values, and the changed values can be placed in the table before any table reference is made.

Tables can be initialized to a common value by a MOVE to the group item that defines the entire table. For example:

```
MOVE SPACES TO TABLE-ONE.
```

However, care should be exercised when this method is used with a table containing non-display type elements. For example:

```
01 BINARY-TABLE.
   05 BINARY-COUNT OCCURS 4 TIMES
      PIC 9999 COMP-4.
   .
   .
   .
   MOVE ZEROS TO BINARY-TABLE.
```

The MOVE statement does not fill BINARY-TABLE with binary zeros, but with display-type zeros, hex “F0”.

The following example shows two ways of initializing a table with zeros:

```
. 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7

                  DATA DIVISION.
                  WORKING-STORAGE SECTION.
                  01  SUB1                  PIC 999.
                  01  TABLE-OF-ELEMENTS.
                      03  ELEMENTS-OF-TABLE  PIC 9     OCCURS 100 TIMES.
                  PROCEDURE DIVISION.
                  100-START-PROCESSING.
                      PERFORM SAMPLE-PERFORM THRU PERFORM-EXIT
                          VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 GREATER THAN 100.
                      PERFORM ANOTHER-WAY-TO-INITIALIZE THRU AWTI-EXIT.
                 *    THE TABLE WILL BE ALL ZEROS AND SHOULD PRINT AS SUCH.
                      DISPLAY "THE TABLE    " TABLE-OF-ELEMENTS.
                      STOP-RUN.
                  SAMPLE-PERFORM.
                      MOVE ZEROS TO ELEMENTS-OF-TABLE (SUB1).
Initializing      PERFORM-EXIT.
Table to              EXIT.
Zeros             ANOTHER-WAY-TO-INITIALIZE.
                      MOVE ZEROS TO TABLE-OF-ELEMENTS.
                  AWTI-EXIT.
                      EXIT.
```

Data Division–Table Handling

COBOL Data Division clauses used for table handling are the OCCURS clause and the USAGE IS INDEX clause.

OCCURS Clause

The OCCURS clause eliminates the need to specify separate entries for repeated data items; it also supplies the information necessary for the use of subscripts or indexes. The formats of the OCCURS clause are as follows:

Format 1–Fixed Length Tables

```
OCCURS integer-2 TIMES
   [ { ASCENDING  } KEY IS data-name-2 [ data-name-3 ] . . . ] . . .
   [ { DESCENDING }                                          ]

   [ INDEXED BY index-name-1 [ index-name-2 ] ... ]
```

Format 2–Variable Length Tables

```
OCCURS integer-1 TO integer-2 TIMES

   DEPENDING ON data-name-1
   [ { ASCENDING  } KEY IS data-name-2 [ data-name-3 ] . . . ] . . .
   [ { DESCENDING }                                          ]

   [ INDEXED BY index-name-1 [ index-name-2 ] ... ]
```

The subject of an OCCURS clause is the data-name of the data item containing the OCCURS clause. Except for the OCCURS clause itself, data description clauses used with the subject apply to each occurrence of the item described.

Whenever the subject is used in any statement–other than SEARCH or USE FOR DEBUGGING, or unless it is the object of a REDEFINES clause–the subject must be subscripted or indexed. When it is subscripted or indexed, the subject refers to one occurrence within the table element.

Whenever the subject is used in a SEARCH or USE FOR DEBUGGING statement, or when it is the object of a REDEFINES clause, the subject must not be subscripted or indexed. When it is not subscripted or indexed, the subject represents the entire table length.

The table must contain less than 32 768 occurrences, the length of a table element must be less than 32 K bytes, and the length of the whole table must be less than 32 K bytes.

All data-names used in the OCCURS clause can be qualified; they cannot be subscripted or indexed.

All integers must be positive nonzero integers.

The OCCURS clause cannot be specified in a data description entry that:

- Has a level-01, level-66, level-77, or level-88 number.
- Describes an item of variable size (an item is of variable size if any subordinate entry contains an OCCURS DEPENDING ON clause).
- Describes redefined data items. (However, a redefined item can be subordinate to an item containing an OCCURS clause.) See “REDEFINES Clause” on page 319.

Fixed Length Tables

When Format 1 is used, integer-2 specifies the exact number of occurrences.

Integer-2 must be greater than zero and less than 32 768.

Because three subscripts or indexes are allowed, three nested levels of the Format 1 OCCURS clause are allowed.

Variable Length Tables

When the OCCURS DEPENDING ON clause is specified, integer-1 represents the minimum number of occurrences, and integer-2 represents the maximum number of occurrences. The value of integer-1 must be one or greater; it must also be less than integer-2. Integer-2 must be less than 32 768. The length of the subject item is fixed; it is only the number of repetitions of the subject item that is variable.

Data-name-1 is the object of the OCCURS DEPENDING ON clause. The object is the data item whose current value represents the current number of occurrences of the subject item. The object of the OCCURS DEPENDING ON clause:

- Must be described as a positive integer. That is, if data-name-1 is described as a signed item, at run time it must contain positive data.

- Must not occupy any storage position within the range of this table. That is, the object must not occupy any storage position from the first character position in this table through the last character position in this record description entry.
- Must contain a value within the range of integer-1 and integer-2, inclusive.

The value of the object of the OCCURS DEPENDING ON clause specifies that part of the table element available to the object program. Items whose occurrence numbers exceed the value of the object are not available. If, during processing, the value of the object is reduced, the contents of items whose occurrence numbers exceed the new value of the object are unpredictable.

When a group item containing a subordinate OCCURS DEPENDING ON item is referred to, the current value of the object determines which part of the table area is used in the operation.

In one record description entry, any entry that contains an OCCURS DEPENDING ON clause may be followed only by items subordinate to it. The OCCURS DEPENDING ON clause cannot be specified as subordinate to another OCCURS clause. However, the Format 1 OCCURS clause may be specified as subordinate to the OCCURS DEPENDING ON clause; in this case, a table of up to three dimensions may be specified.

ASCENDING/DESCENDING KEY Phrase

The ASCENDING/DESCENDING KEY phrase specifies that the repeated data is arranged in ascending or descending key sequence (depending on the keyword specified) according to the values contained in data-name-2, data-name-3, and so on. The data-names are listed in their descending order of significance. The ASCENDING/DESCENDING KEY data items are used by the SEARCH ALL statement for a search of the table element.

The order is determined by the rules for comparison of operands. (See “Simple Conditions” on page 354, “Relation Condition” on page 356.)

Data-name-2 must be the name of the subject entry or the name of an entry subordinate to the subject entry. If data-name-2 names the subject entry, that entire entry becomes the ASCENDING/DESCENDING KEY and is the only key that can be specified for this table element. If data-name-2 does not name the subject entry, then data-name-2, data-name-3, and so on:

- Must be subordinate to the subject of the table entry itself
- Must not be subordinate to any other entry that contains an OCCURS clause
- Must not themselves contain an OCCURS clause.

The following example illustrates the specification of ASCENDING/DESCENDING KEY data items:

```
WORKING-STORAGE SECTION.
01 CURRENT-WEEK PICTURE 99.
01 TABLE-RECORD.
   05 EMPLOYEE-TABLE OCCURS 100 TIMES
      ASCENDING KEY IS WAGE-RATE
      EMPLOYEE-NO INDEXED BY A, B.
      10 EMPLOYEE-NAME PIC X(20).
      10 EMPLOYEE-NO PIC 9(6).
      10 WAGE-RATE PIC 9999V99.
      10 WEEK-RECORD OCCURS 52 TIMES
         ASCENDING KEY IS WEEK-NO
         INDEXED BY C.
         15 WEEK-NO PIC 99.
         15 AUTHORIZED-ABSENCES PIC 9.
         15 UNAUTHORIZED-ABSENCES PIC 9.
         15 LATENESSES PIC 9.
```

The keys for EMPLOYEE-TABLE are subordinate to that entry, and the key for WEEK-RECORD is subordinate to that subordinate entry.

When the ASCENDING/DESCENDING KEY phrase is specified, the following rules apply:

- Keys must be listed in decreasing order of significance.
- A key can have USAGE DISPLAY or COMPUTATIONAL.

IBM Extension

A key can have USAGE COMPUTATIONAL-3 or COMPUTATIONAL-4.

End of IBM Extension

- The user is responsible for ensuring that the data present in the table is arranged in ascending or descending key sequence according to the collating sequence in use.

In the preceding example, records in EMPLOYEE-TABLE must be arranged in ascending order of WAGE-RATE and in ascending order of EMPLOYEE-NO within WAGE-RATE. Records in Week-Record must be arranged in ascending order of WEEK-NO. If they are not, SEARCH ALL statement results will be unpredictable.

INDEXED BY Phrase

The INDEXED BY phrase specifies the indexes that can be used with this table element. The INDEXED BY phrase is required if indexing is used to refer to this table element.

Each index-name must follow the rules for formation of a user-defined word; at least one character must be alphabetic. Each index-name specifies an index to be created by the compiler for use by the program. These index-names are not data-names and are not identified elsewhere in the COBOL program; instead, they can be regarded as compiler generated registers for the use of this object program only. Therefore, they are not data or part of any data hierarchy; as such, each must be unique. An INDEX-NAME can only be referenced by a PERFORM, SET, or SEARCH state-

ment, as a parameter in the USING phrase in a CALL statement, or in a relational condition comparison.

USAGE IS INDEX Clause

The USAGE IS INDEX clause specifies that the data item named has an index format. Such an item is an index data item.

Format

```
[ USAGE IS ] INDEX
```

An index data item is a two-byte elementary item that can be used to save index-name values for future reference. Through the SET statement, an index data item can be assigned an index-name value. The index-name value corresponds to the displacement for an occurrence number in the table, that is (occurrence-number - 1) * entry length.

An index data item can be referred to directly only in a SEARCH statement, a SET statement, a relation condition, the USING phrase of the Procedure Division header, or the USING phrase of the CALL statement. An index data item can be part of a group item referred to in a MOVE statement or an input/output statement.

An index data item saves binary values that represent a table occurrence number; however, it is not itself necessarily defined as part of any table. Thus, when it is referenced directly in a SEARCH or SET statement, or indirectly in a MOVE or input/output statement, there is no conversion of values when the statement is processed.

The USAGE IS INDEX clause may be written at any level. If a group item is described with the USAGE IS INDEX clause, it is the elementary items within the group that are index data items; the group itself is not an index data item, and the group name cannot be used in SEARCH and SET statements or in relation conditions. The USAGE clause of an elementary item cannot contradict the USAGE clause of a group to which the item belongs.

An index data item cannot be a conditional variable; it cannot have a subordinate level-88 item.

The SYNCHRONIZED, JUSTIFIED, PICTURE, BLANK WHEN ZERO, or VALUE clauses cannot be used to describe group or elementary items described with the USAGE IS INDEX clause.

Since the format of an index data item is implementation dependent, an index data item should not be defined in the File Section.

Procedure Division–Table Handling

In the Procedure Division, the SEARCH and SET statements can be specified with indexed tables. There are also special rules involving table handling elements when they are used in relation conditions.

Relation Conditions

Comparisons involving index-names and/or index data items conform to the following rules:

- The comparison of two index-names is actually the comparison of the corresponding occurrence numbers.
- In the comparison of an index-name with a data item (other than an index data item) or in the comparison of an index-name with a literal, the occurrence number that corresponds to the value of the index-name is compared with the data item or literal.
- In the comparison of an index data item with an index-name or another index data item, the actual values are compared without conversion. Results of any other comparison involving an index data item are undefined.

Figure 92 shows permissible comparisons for index-names and index data items.

	SECOND OPERAND			
First Operand	Index-name	Index Date Item	Data-Name (numeric integer only)	Numeric Literal (integer only)
Index-Name	Compare occurrence numbers	Compare without conversion	Compare occurrence number with data-name	Compare occurrence number with literal
Index Data Item	Compare without conversion	Compare without conversion	Invalid	Invalid
Data-Name (numeric integer only)	Compare occurrence number with data-name	Invalid	Invalid	Invalid
Numeric Literal (integer only)	Compare occurrence number with literal	Invalid	Invalid	Invalid

Figure 92. Permissible Comparisons for Index-Names and Index Data Items

SEARCH Statement

The SEARCH statement searches a table for an element that satisfies the specified condition, and adjusts the associated index to indicate that element. The formats for the SEARCH statement are:

Format 1

```
SEARCH identifier-1 [ VARYING { identifier-2 } ] [ AT END imperative-statement-1 ]
                              { index-name-1 }

     WHEN condition-1 { imperative-statement-2 }
                      { NEXT SENTENCE          }

   [ WHEN condition-2 { imperative-statement-3 } ] . . .
                      { NEXT SENTENCE          }
```

Format 2

```
SEARCH ALL indentifier-1 [ AT END imperative-statement-1 ]

        { data-name-1 { IS EQUAL TO } { identifier-3            } }
        {             { IS =        } { literal-1               } }
   WHEN {                            { arithmetic-expression-1 } }
        { condition-name-1                                       }

        [                                                        ]
        [     { data-name-2 { IS EQUAL TO } { identifier-4            } } ]
        [     {             { IS =        } { literal-2               } } ]
        [ AND {                           { arithmetic-expression-2 } } ] . . .
        [     {                                                       } ]
        [     { condition-name-2                                      } ]
        [                                                        ]

        { imperative-statement-2 }
        { NEXT SENTENCE          }
```

The Data Division description of identifier-1 must contain an OCCURS clause with the INDEXED BY phrase.

When specified in the SEARCH statement, identifier-1 must refer to all occurrences within the table element; it must not be subscripted or indexed.

Identifier-1 can be a data item subordinate to a data item that contains an OCCURS clause; it can be a part of a two- or three-dimensional table. In this case, the data description entry must specify an INDEXED BY phrase for each dimension of the table.

SEARCH statement processing modifies only the value in the index-name associated with identifier-1 (and, if present, of index-name-1 or identifier-2). Therefore, to search an entire two- or three-dimensional table, a SEARCH statement must be processed for each dimension. Before each processing, SET statements must be processed to reinitialize the associated index-names.

In the AT END and WHEN phrases, control passes to the next sentence after the imperative-statement is processed if any of the specified imperative-statements do not end with a GO TO statement.

Format 1

Format 1 SEARCH statement processing causes a serial search, beginning at the current index setting.

If the value of the index-name associated with identifier-1 is not greater than the highest possible occurrence number, when the search begins the following actions take place:

1. The conditions in the WHEN phrases are evaluated in the order they are written.
2. If none of the conditions are satisfied, the index-name for identifier-1 is incremented to correspond to the next table element, and step 1 is repeated.
3. If upon evaluation, one of the WHEN conditions is satisfied, the search terminates immediately, and the imperative-statement associated with that condition is processed. The index-name identifies the table element that satisfied the condition.

4. If the end of the table is reached (that is, the incremented index-name value is greater than the highest possible occurrence number) without the WHEN condition begin satisfied, the search terminates as described in the next paragraph.

If, when the search begins, the value of the index-name associated with identifier-1 is greater than the highest possible occurrence number, the search immediately ends, and, if specified, the AT END imperative-statement is processed. If the AT END phrase is omitted, control passes to the next sentence.

Each WHEN phrase condition can be any condition as described under in "Conditional Expressions" on page 354.

VARYING Index-Name-1 Phrase: When the VARYING index-name-1 phrase is omitted, the first (or only) index-name for identifier-1 is used for the search. When the VARYING index-name-1 phrase is specified, one of the following actions takes place:

- If index-name-1 is an index for identifier-1, this index is used for the search. Otherwise, the first (or only) index-name is used.
- If index-name-1 is an index for another table element, then the first (or only) index-name for identifier-1 is used for the search; the occurrence number represented by index-name-1 is incremented by the same amount as the search index-name and at the same time.

VARYING Identifier-2 Phrase: When this phrase is specified, the first (or only) index-name for identifier-1 is used for the search.

Identifier-2 must be either an index data item or an elementary integer item. During the search, one of the following actions takes place:

- If identifier-2 is an index data item, then whenever the search index is incremented, the specified index item is simultaneously incremented by the same amount.
- If identifier-2 is an elementary integer item, then whenever the search index is incremented, the specified data item is simultaneously incremented by one.

Figure 93 on page 485 is a flowchart of a Format 1 SEARCH operation containing two WHEN phrases.

Format 2

Format 2 SEARCH ALL statement processing causes a binary search to be made. The search index need not be initialized by SET statements, because its setting is varied during the search operation. The index used is always the index that is associated with the first index-name specified in the OCCURS clause.

If the WHEN phrase cannot be satisfied for any setting of the index within this range, the search is unsuccessful. If the AT END phrase is specified, the AT END imperative-statement is processed. If the AT END phrase is not specified, control is passed to the next sentence. In either case, the final setting of the index is not predictable.

If the WHEN phrase can be satisfied, control passes to imperative-statement-2 and the index contains a value indicating an occurrence that allows the WHEN condition(s) to be satisfied.

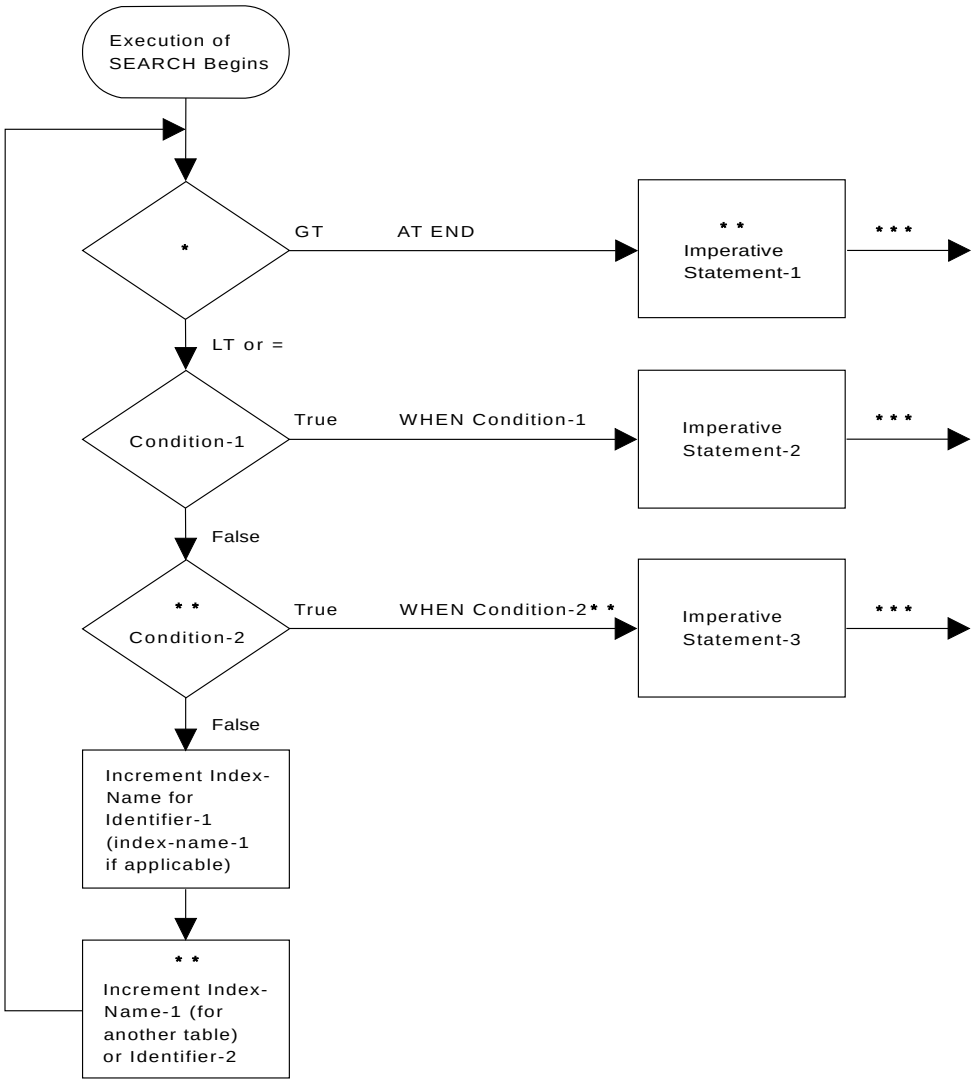
WHEN Condition-Name Phrase: If the WHEN condition-name phrase is specified, each condition-name specified must have only a single value, and each must be associated with an ASCENDING/DESCENDING KEY identifier for this table element.

WHEN Relation-Condition Phrase: If WHEN relation-condition is specified, the following considerations apply:

- Data-name-1 or data-name-2 must specify an ASCENDING/DESCENDING KEY data item in the identifier-1 table element and must be indexed by the first identifier-1 index-name, along with other indexes or literals as required. Each data-name may be qualified.
- Identifier-3 and identifier-4 must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index-name for identifier-1.
- Literal-1 or literal-2 must be a positive or unsigned numeric integer.
- Arithmetic-expression-1 or arithmetic-expression-2 may be any of those defined under “Arithmetic Expressions” on page 352 with the following restriction: any identifier in the arithmetic-expression must not be an ASCENDING/DESCENDING KEY data item for identifier-1 or an item that is indexed by the first index-name for identifier-1.
- When an ASCENDING/DESCENDING KEY data item is specified either explicitly or implicitly in the WHEN phrase, then all preceding ASCENDING/DESCENDING KEY data-names for identifier-1 must also be specified.

The results of a SEARCH ALL operation are predictable only when both of the following apply:

- The data in the table is ordered in ascending or descending key sequence.
- The contents of the ASCENDING/DESCENDING keys specified in the WHEN phrase provide a unique table reference.



* Index setting equals highest permissible occurrence number.

** These operations are included only when called for in the statement.

*** Each of these control transfers is to the next sentence unless the imperative-statement ends with a GO TO statement.

Figure 93. Format 1 SEARCH with Two WHEN Phrases

Programming Notes

Index data items cannot be used as subscripts or indexes, because of the restrictions on direct reference to them. The use of a direct indexing reference together with a relative indexing reference for the same index-name allows reference to two different occurrences of a table element for comparison purposes.

When the object of the VARYING phrase is an index-name for another table element, one Format 1 SEARCH statement looks at two table elements at once.

One Format 4 PERFORM statement can search an entire multidimensional table.

To ensure correct processing of a PERFORM or SEARCH statement for a variable length table, the user must make sure that the object of the OCCURS DEPENDING ON clause (data-name-1) contains a value that correctly specifies the current length of the table.

SEARCH Example

The following example searches an inventory table for items that match those from input data. The key is ITEM-NUMBER.

```
.. 1 ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ... 7

 DATA DIVISION.
 FILE SECTION.
 FD  SALES-DATA
     BLOCK CONTAINS 1 RECORDS
     RECORD CONTAINS 80 CHARACTERS
     LABEL RECORDS STANDARD
     DATA RECORD IS SALES-REPORTS.
 01  SALES-REPORTS          PIC X(80).
 FD  PRINTED-REPORT
     BLOCK CONTAINS 1 RECORDS
     RECORD CONTAINS 132 CHARACTERS
     LABEL RECORDS OMITTED
     DATA RECORD IS PRINTER-OUTPUT.
 01  PRINTER-OUTPUT         PIC X(132).
 FD  INVENTORY-DATA
     BLOCK CONTAINS 1 RECORDS
     RECORD CONTAINS 40 CHARACTERS
     LABEL RECORDS STANDARD
     DATA RECORD IS INVENTORY-RECORD.
 01  INVENTORY-RECORD.
     03  I-NUMBER           PIC 9(4).
     03  INV-ID             PIC X(26).
     03  I-COST             PIC 9(8)V99.
 WORKING-STORAGE SECTION.
 01  EOF-SW                 PIC X      VALUE "N".
 01  EOF-SW2                PIC X      VALUE "N".
 01  SUB1                   PIC 99.
 01  RECORDS-NOT-FOUND      PIC 9(5)    VALUE ZEROS.
 01  TOTAL-COSTS            PIC 9(10)   VALUE ZEROS.
 01  HOLD-INPUT-DATA.
     03  INVENTORY-NUMBER  PIC 9999.
     03  PURCHASE-COST     PIC 9(4)V99.
     03  PURCHASE-DATE     PIC 9(6).
     03  FILLER            PIC X(64).
 01  PRINTER-SPECS.
     03  PRINT-LINE.
         05  OUTPUT-ITEM-NUMBER        PIC ZZZ9.
         05  FILLER                    PIC X(48) VALUE SPACES.
         05  TOTAL-COSTS-0             PIC $(8).99.
 01  PRODUCT-TABLE.
     05  INVENTORY-NUMBERS  OCCURS 50 TIMES
                            ASCENDING KEY ITEM-NUMBER
                            INDEXED BY INDEX-1.
             07  ITEM-NUMBER           PIC 9(4).
             07  ITEM-DESCRIPTION      PIC X(26).
             07  ITEM-COST             PIC 9(8)V99.
```

```
.. 1  ... ... 2 ... ... 3 ... ... 4 ... ... 5 ... ... 6 ... ...7

 PROCEDURE DIVISION.
 100-START-IT.
     OPEN INPUT SALES-DATA INVENTORY-DATA OUTPUT PRINTED-REPORT.
     MOVE HIGH-VALUES TO PRODUCT-TABLE.
     PERFORM READ-INVENTORY-DATA.
 LOAD-TABLE-ROUTINE.
     PERFORM LOAD-IT VARYING SUB1 FROM 1 BY 1 UNTIL SUB1 > 50
           OR EOF-SW2 = "Y".
     PERFORM 110-READ-IT.
 200-MAIN-ROUTINE.
     PERFORM PROCESS-DATA UNTIL EOF-SW = "Y".
     MOVE TOTAL-COSTS TO TOTAL-COSTS-0.
     PERFORM WRITE-REPORT THRU WRITE-REPORT-EXIT.
     DISPLAY "RECORDS NOT FOUND - " RECORDS-NOT-FOUND.
     STOP RUN.
 PROCESS-DATA.
     SEARCH ALL INVENTORY-NUMBERS
       AT END PERFORM KEY-NOT-FOUND THRU NOT-FOUND-EXIT
       WHEN ITEM-NUMBER (INDEX-1) = INVENTORY-NUMBER
           MOVE ITEM-NUMBER (INDEX-1) TO OUTPUT-ITEM-NUMBER
           MOVE ITEM-COST (INDEX-1) TO TOTAL-COSTS-0
           ADD ITEM-COST (INDEX-1) TO TOTAL-COSTS
           PERFORM WRITE-REPORT THRU WRITE-REPORT-EXIT.
     PERFORM 110-READ-IT.
 KEY-NOT-FOUND.
     ADD 1 TO RECORDS-NOT-FOUND.
 NOT-FOUND-EXIT.
     EXIT.
 LOAD-IT.
     MOVE INVENTORY-RECORD TO INVENTORY-NUMBERS (SUB1).
     PERFORM READ-INVENTORY-DATA.
 WRITE-REPORT.
     WRITE PRINTER-OUTPUT FROM PRINTER-SPECS.
 WRITE-REPORT-EXIT.
     EXIT.
*********END OF MAIN PROGRAM********
 READ-INVENTORY-DATA.
     READ INVENTORY-DATA
       AT END MOVE "Y" TO EOF-SW2.
 110-READ-IT.
     READ SALES-DATA INTO HOLD-INPUT-DATA
       AT END MOVE "Y" TO EOF-SW.
*********END OF EXAMPLE SEARCH PROGRAM********
```

SET Statement

The SET statement establishes reference points for table handling operations by setting index-names to values associated with table elements. The SET statement may be used to transfer values between index-names and other elementary data items. The formats of the SET statement when it is used for table handling are described here. For information on the other formats allowed for SET statements, see “SET Statement” on page 440.

Index-names are related to a given table through the INDEXED BY phrase of the OCCURS clause; they are not further defined in the program.

When the sending and receiving fields in a SET statement share part of their storage (that is, the operands overlap), the result of the processing of such a SET statement is undefined.

Format 3

Format 3

```
SET { identifier-1 [ , identifier-2 ] . . . } TO { identifier-3 }
    { index-name-1 [ , index-name-2 ] . . . }    { index-name-3 }
                                                 { integer-1    }
```

When this form of the SET statement is processed, the value of the sending field replaces (with or without conversion) the current value of the receiving field.

The receiving field can be specified as follows:

- Index-name-1, index-name-2, and so on.
- Identifier-1, identifier-2, and so on. The identifiers must name either index data items or elementary numeric integer items.

The sending field can be specified as follows:

- Identifier-3, which must name either an index data item or an elementary numeric integer item
- Index-name-3, whose value before the SET statement is processed must correspond to an occurrence number of its associated table
- Integer-1, which must be a positive integer.

Figure 94 on page 490 shows valid combinations of sending and receiving fields in a Format 3 SET statement.

Processing of the Format 3 SET statement depends upon the type of receiving field, as follows:

- Index-name receiving fields (index-name-1, index-name-2, and so on) with one exception are converted to a displacement value representing the occurrence number indicated by the sending field. To be valid, the resulting index-name value must correspond to an occurrence number in its associated table element. For the one exception, when the sending field is an index data item, the value in the index data item is placed in the index-name without change.
- Index data item receiving fields (identifier-1, identifier-2, and so on) are set equal to the contents of the sending field (which must be either an index-name or an index data item); no conversion takes place. A numeric integer or literal sending field must not be specified.
- Integer data item receiving fields (identifier-1, identifier-2, and so on) are set to the occurrence number associated with the sending field, which must be an index-name. An integer data item, an index data item, or a literal sending field must not be specified.

Receiving fields are acted upon in the left-to-right order they are specified. Any subscripting or indexing associated with an identifier receiving field is evaluated immediately before the field is acted upon.

The value used for the sending field is its value at the beginning of SET statement processing.

The value for an index-name after processing of a SEARCH or PERFORM statement can be undefined; therefore, a Format 3 SET statement should be used to reinitialize such index-names before other table handling operations are attempted.

Format 4

Format 4

```
SET index-name-4 [ , index-name-5 ] . . . { UP BY   } { identifier-4 }
                                          { DOWN BY } { integer-2    }
```

When this form of the SET statement is processed, the value of the receiving field is incremented (UP BY) or decremented (DOWN BY) by the value in the sending field.

The receiving field can be specified by index-name-4, index-name-5, and so on. These index-name values must correspond to an occurrence number in the associated table both before and after the SET statement processing.

The sending field can be specified as identifier-4, which must be an elementary integer data item, or as integer-2, which must be an integer.

When the Format 4 SET statement is processed, the contents of the receiving field are incremented (UP BY) or decremented (DOWN BY) by the value of identifier-4 or integer-2. Receiving fields are acted upon in the left-to-right order they are specified. The value of the sending field at the beginning of SET statement processing is used for all receiving fields.

	RECEIVING FIELD		
SENDING FIELD	Index-name	Index Data Item	Integer Data Item
Index-Name	Valid	Valid*	Valid
Index data item	Valid*	Valid	
Integer data item	Valid		
Integer literal	Valid		
* No conversion takes place.			

Figure 94. Sending and Receiving Fields for Format 3 SET Statements

SORT/MERGE

Arranging records in a particular order or sequence is a common requirement in data processing; such record ordering can be accomplished using sort or merge operations. While both operations accomplish record ordering, the functions and capabilities of a sort and a merge are different.

A sort produces an ordered file from one or more input files that can be completely unordered as to sort sequence. Thus, the sort operation must accept unordered input and produce ordered output.

A merge produces an ordered file from two or more input files, each of which is already ordered in the merge sequence.

IBM Extension

Input files need not be sequenced prior to a merge operation.

End of IBM Extension

COBOL has special language features that assist in sort and merge operations so that the user need not program these operations in detail.

Sort/Merge Concepts

Sorting and merging have always constituted a large percentage of the workload in business data processing. COBOL standardizes the specification of these operations, making them easy to specify and modify. In addition, the COBOL user can alternatively use the AS/400 logical file support to process these operations as separate command language (CL) commands. The COBOL language supports these operations through the file-control entry in the Environment Division, the SD (sort-merge-file-description) entry in the Data Division, and the SORT and MERGE statements in the Procedure Division.

The sort or merge file is described through the file-control entry in the Environment Division, and the SD entry in the Data Division. The sort or merge file is the working file used during the sort or merge; it can be considered an internal file. As such, blocking and internal storage allocation for this file are not under the control of the COBOL user. However, a sort or merge file, like any file, is a set of records, and a sort-merge file description can be considered a particular type of file description.

The sort-merge file is processed through a Procedure Division SORT or MERGE statement. The statement specifies the key field(s) within the record upon which the sort or merge is to be arranged. Keys can be specified as ascending or descending. When more than one key is specified, a mixture of the two sequences is allowed. The sequence of sorted or merged records conforms to the mixture of keys specified.

Sort Concepts

Through the SORT statement, the COBOL user has access to input procedures (used before sorting) and output procedures (used after sorting) that can add, delete, alter, edit, or otherwise modify the records in the input and/or output files. A COBOL program can contain any number of sorts, each with its own independent input and/or output procedures. During SORT statement processing, these procedures are automatically run at the specified point in processing; thus, extra passes through the sort file are avoided.

A COBOL program containing a sort is usually organized so that one or more input files are read and operated on by an input procedure. Within the input procedure a RELEASE statement (analogous to the WRITE statement) places a record in the sort file. That is, when input procedure processing is completed, a sort file has been created by placing records one at a time into the sort file through the RELEASE statement. If the user does not wish to modify the records before the sorting operation begins, the SORT statement USING phrase releases the unmodified records to the sort file.

After all the input records have been placed in the sort file, the sorting operation is run. This operation arranges the entire set of sort file records in the sequence specified by the key(s).

After completion of the sorting operation, sorted records can be made available from the sort file, one at a time, through a RETURN statement for modification in an output procedure. If the user does not wish to modify the sorted records, the SORT statement GIVING option names the sorted output file.

Merge Concepts

Through the MERGE statement, the COBOL user has access to output procedures (used after merging) that can modify the records in the output file. The COBOL program can contain any number of merge operations, each with its own independent output procedures. During MERGE statement processing, these procedures are automatically run at the specified point in processing.

The merge operation compares keys within the records of the input files and arranges the records within the merged file in the sequence specified by the key(s).

Merged records can then be made available, one at a time, through a RETURN statement for modification in an output procedure. If the user does not wish to modify the merged records, the MERGE statement GIVING phrase names the merged output file.

Environment Division–SORT/MERGE

In the Environment Division, the user must write file-control entries for each file used as input to or output from a sort or merge operation. The user must also write a file-control entry for each unique sort-file or merge-file.

File-Control Paragraph

See “FILE-CONTROL Paragraph” on page 281 for a description of input and output files of a sort or merge operation.

I-O-Control Paragraph

In the I-O-Control paragraph, the SAME SORT AREA or SAME SORT-MERGE AREA clause is used.

Format

```
[ [ SAME [ RECORD    ] ] AREA FOR file-name-2 { , file-name-3 } . . . ] . . .
         [ SORT      ]
         [ SORT-MERGE ]
```

The SAME SORT AREA and SAME SORT-MERGE AREA clauses are syntax-checked, but are treated as documentation.

Restrictions on the specification of SAME RECORD AREA clause are given under “I-O-CONTROL Paragraph” on page 292.

Data Division–SORT/MERGE

In the File Section, the user must write an FD entry for each file that is input to or output from the sort/merge operation, as well as a record description entry. In addition, there must be an SD (sort-merge-file-description) entry for each sort or merge file.

Format 5–Sort or Merge File Description

```
[ SD file-name

   [ RECORD CONTAINS [ integer-1 TO ] integer-2 CHARACTERS ]

   [ DATA { RECORD IS   } data-name-1 [ , data-name-2 ] . . . ] .
          { RECORDS ARE }

{ record-description-entry } . . . ]
```

The level indicator SD identifies the beginning of the SD entry, and must precede the file-name. The file-name must specify a sort or merge file.

The clauses that follow file-name are optional, and their order of appearance is not significant. Both the RECORD CONTAINS clause and the DATA RECORDS clause are described in Chapter 9, “Data Division”

One or more record description entries must follow the SD entry. However, no input/output statements may be processed for this file.

The following example illustrates the File Section entries needed for a sort or merge file:

```
SD SORT-FILE.
01 SORT-RECORD PICTURE X(80).
```

Procedure Division–SORT/MERGE

The Procedure Division contains MERGE and SORT statements to describe the merge and sort operations and, optionally, sort input procedures and/or sort/merge output procedures. A sort input procedure must contain a RELEASE statement that makes each record available to the sorting operation. A sort/merge output procedure must contain a RETURN statement that makes a sorted/merged record available to the output procedure.

The Procedure Division can contain more than one SORT and/or MERGE statement. These statements can appear anywhere except in the Declaratives portion or in the sort input or sort/merge output procedures.

Files specified in the USING and GIVING phrases of the SORT and MERGE statements must be described explicitly or implicitly in their file-control entries as having sequential organization.

USE procedures are not run if they reference files specified on a USING or GIVING phrase of a SORT or MERGE statement. If these files are also referenced in an I-O statement within an output procedure or a SORT input procedure, a USE procedure for the file specified is called when necessary.

MERGE Statement

The MERGE statement combines two or more identically sequenced files that have already been sorted in an identical ascending/descending key sequence on one or more keys. This statement makes records available in merged order to an output procedure or output file.

Format

```
MERGE file-name-1 ON { ASCENDING  } KEY data-name-1 [ , data-name-2 ] . . .
                     { DESCENDING }

             [ ON { ASCENDING  } KEY data-name-3 [ , data-name-4 ] . . . ] . . .
                { DESCENDING }

    [ COLLATING SEQUENCE IS alphabet-name ]

      USING file-name-2, file-name-3 [ , file-name-4 ] . . .

    {                                                                    }
    { OUTPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ]  }
    {                                    [ { THRU    }                ]  }
    {                                                                    }
    { GIVING file-name-5                                                 }
```

File-name-1 is the name given in the SD entry that describes the records being merged. No file-name may be repeated in the MERGE statement.

When the MERGE statement is processed, all records contained in file-name-2, file-name-3, and so on, are accepted by the sort/merge program and then merged according to the key(s) specified. These files must not be open when the MERGE statement is processed; they are automatically opened and closed by the MERGE operation, and all implicit functions are processed. The files are closed as if the CLOSE statement were written without any optional processing.

See “MERGE Statement and SORT Statement Phrases” for details about the phrases of the MERGE statement.

SORT Statement

The SORT statement accepts records from one or more files, sorts them according to the specified key(s), and makes records available either through an output procedure or in an output file.

Format

```
SORT file-name-1 ON { ASCENDING  } KEY data-name-1 [ , data-name-2 ] . . .
                    { DESCENDING }

             [ ON { ASCENDING  } KEY data-name-3 [ , data-name-4 ] . . . ] . . .
                { DESCENDING }

   [ COLLATING SEQUENCE IS alphabet-name ]

   {                                                                    }
   { INPUT PROCEDURE IS section-name-1 [ { THROUGH } section-name-2 ]   }
   {                                   [ { THRU    }                ]   }
   {                                                                    }
   {                                                                    }
   { USING file-name-2 [ , file-name-3 ] . . .                          }

   {                                                                    }
   { OUTPUT PROCEDURE IS section-name-3 [ { THROUGH } section-name-4 ]  }
   {                                    [ { THRU    }                ]  }
   {                                                                    }
   { GIVING file-name-4                                                 }
```

File-name-1 is the name given in the SD entry that describes the records being sorted.

When the SORT statement is processed, all records contained in file-name-2, file-name-3, and so on are accepted by the sort/merge program and then sorted according to the key(s) specified. These input files must not be open at the time the SORT statement is processed; they are automatically opened and closed by the SORT operation, and all implicit functions are processed. The files are closed as if the CLOSE statement were written without any optional processing.

MERGE Statement and SORT Statement Phrases

Most SORT/MERGE statement phrases apply to both the SORT and the MERGE statements. The common SORT/MERGE statement phrases are the ASCENDING/DESCENDING KEY phrase, the COLLATING SEQUENCE phrase, the USING phrase, the GIVING phrase, and the OUTPUT PROCEDURE phrase. The INPUT PROCEDURE phrase applies only to the SORT statements.

ASCENDING/DESCENDING KEY Phrase

This phrase specifies that records are to be processed in an ascending or descending key sequence based on the specified sort/merge keys.

Each data-name specifies a KEY data item on which the sort-merge will be based. Each such data-name must identify a data item in a record associated with file-name-1. The following rules apply:

- A specific KEY data item must be physically located in the same position and have the same data format in each input file; however, it need not have the same data-name.
- If file-name-1 has more than one record description, then the KEY data items need be described in only one of the record descriptions.
- KEY data items must be fixed-length items.
- KEY data items must not contain an OCCURS clause or be subordinate to an item that contains an OCCURS clause.
- The total length (in bytes) of the KEY data items must not exceed 248.
- KEY data items can be qualified; they cannot be subscripted or indexed.

The KEY data items are listed in order of decreasing significance, regardless of how they are divided into KEY phrases. Using the SORT format as an example, data-name-1 is the most significant key and records are processed in ascending or descending order on that key; data-name-2 is the next most significant key and within data-name-1 records are processed on data-name-2 in ascending or descending order. Within data-name-2, records are processed on data-name-3 in ascending or descending order; within data-name-3, records are processed on data-name-4 in ascending or descending key sequence.

The direction of the sort/merge operation depends on the specification of the ASCENDING or DESCENDING keywords as follows:

- When ASCENDING is specified, the sequence is from the lowest key value to the highest key value.
- When DESCENDING is specified, the sequence is from the highest key value to the lowest.
- If the KEY data item is alphabetic, alphanumeric, alphanumeric edited, or numeric edited, the sequence of key values depends on the collating sequence used.
- The key comparisons are processed according to the rules for comparison of operands in a relation condition. See “Simple Conditions” on page 354, “Relation Condition” on page 356.

COLLATING SEQUENCE Phrase

This phrase specifies the collating sequence to be used in nonnumeric comparisons for the KEY data items in this sort/merge operation.

Alphabet-name must be specified in the SPECIAL-NAMES paragraph alphabet-name clause. Any one of the alphabet-name clause phrases can be specified with the following results:

- When NATIVE is specified, the EBCDIC collating sequence is used for all nonnumeric comparisons.

- When STANDARD-1 is specified, the ASCII collating sequence is used for all nonnumeric comparisons.
- When the literal phrase is specified, the collating sequence established by the specification of literals in the alphabet-name clause is used for all nonnumeric comparisons.

When the COLLATING SEQUENCE phrase is omitted, the PROGRAM COLLATING SEQUENCE clause (if specified) in the OBJECT-COMPUTER paragraph specifies the collating sequence to be used. When both the COLLATING SEQUENCE phrase and the PROGRAM COLLATING SEQUENCE clause are omitted, the EBCDIC collating sequence is used.

USING Phrase

When the USING phrase is specified, all input files are transferred automatically to file-name-1. At the time the SORT or MERGE statement is processed, these files must not be open; the COBOL compiler opens, reads, makes records available, and closes these files automatically.

The input files must have sequential organization.

All input files must be described in an FD entry in the Data Division, and their record descriptions must describe records of the same size as the record described for the sort or merge file. If the elementary items that make up these records are not identical, the user must describe the input records as having the same number of character positions as the sort record.

GIVING Phrase

When the GIVING phrase is specified, all the sorted or merged records in file-name-1 are automatically transferred to the output file (MERGE file-name-5 or SORT file-name-4). At the time the SORT or MERGE statement is processed, this file must not be open; the COBOL compiler opens, writes, and closes the output file automatically. The records overwrite the previous contents, if any, of the file.

IBM Extension

If file-name-1 is a logical data base file, the records are added to the end of the file.

End of IBM Extension

The output file must have sequential organization.

The output file must be described in an FD entry in the Data Division, and its record description(s) must describe records of the same size as the record described for the sort or merge file. If the elementary items that make up these records are not identical, the user must describe the output record as having the same number of character positions as the sort or merge record.

SORT INPUT PROCEDURE Phrase

This phrase specifies the section-name(s) of a procedure that is to modify input records before the sorting operation begins.

Section-name-1 specifies the first (or only) section in the input procedure. Section-name-2 (when specified) identifies the last section of the input procedure.

The input procedure must consist of one or more sections that are written consecutively and do not form a part of any output procedure. The input procedure must include at least one RELEASE statement in order to transfer records to the sort-file.

Control must not be passed to the input procedure except when a related SORT statement is being processed because the RELEASE statement in the input procedure has no meaning unless it is controlled by a SORT statement. The input procedure can include any procedures needed to select, create, or modify records. The following restrictions apply to the procedural statements within an input procedure:

- The input procedure must not contain any SORT or MERGE statements.
- The input procedure must not contain any transfers of control to points outside the input procedure. The processing of a CALL statement to another program, or the processing of USE Declaratives is not considered a transfer of control outside an input procedure. Hence, they are allowed to be activated within these procedures.

IBM Extension

- If control transfers via a PERFORM statement to a point outside the input procedure, a conditional level message is issued but compilation continues.

End of IBM Extension

- The remainder of the Procedure Division must not contain any transfers of control to points inside the input procedure with the exception of the return of control from a Declaratives Section.

IBM Extension

- If control transfers via a PERFORM statement to a point inside the input procedure from elsewhere in the Procedure Division, a conditional level message is issued but the compilation continues.

End of IBM Extension

If an input procedure is specified, control is passed to the input procedure when the SORT program input phase is ready to receive the first record. The compiler inserts a return mechanism at the end of the last section of the input procedure and when control passes the last statement in the input procedure, the records that have been released to file-name-1 are sorted. The RELEASE statement transfers records from the Input Procedure to the sort file, which is then used in the input phase of the sort operation.

SORT/MERGE OUTPUT PROCEDURE Phrase

This phrase specifies the section-name(s) of a procedure that is to modify output records from the sort or merge operation.

Section-name-3 specifies the first (or only) section in the output procedure. Section-name-4 (when specified) identifies the last section of the output procedure.

The output procedure must consist of one or more sections that are written consecutively and are not part of any input procedure. The output procedure must include at least one RETURN statement in order to make sorted/merged records available for processing.

When all the records are sorted/merged, control is passed to the output procedure. The RETURN statement in the output procedure is a request for the next record.

Control must not be passed to the output procedure except when a related SORT or MERGE statement is being processed because RETURN statements in the output procedure have no meaning unless they are controlled by a SORT or MERGE statement. The output procedure can consist of any procedures needed to select, modify, or copy the records that are being returned one at a time from the sort/merge file. There are three restrictions on the procedural statements within the output procedure:

- The output procedure must not contain any SORT or MERGE statements.
- The output procedure must not contain any transfers of control to points outside the output procedure. The processing of a CALL statement to another program, or the processing of USE Declaratives are not considered as transfers of control outside an output procedure. Hence, they are allowed to be activated within these procedures.

IBM Extension

- If control transfers via a PERFORM statement to a point outside the output procedure, a conditional level message is issued but compilation continues.

End of IBM Extension

- The remainder of the Procedure Division must not contain any transfers of control to points inside the output procedure with the exception of the return of control from a Declaratives Section.

IBM Extension

- If control transfers via a PERFORM statement to a point inside the output procedure from elsewhere in the Procedure Division, a conditional level message is issued but the compilation continues.

End of IBM Extension

When an output procedure is specified, control passes to it after the sort/merge file (file-name-1) has been placed in sequence by the sort/merge operation. The COBOL compiler inserts a return mechanism at the end of the last section in the

output procedure; when control is passed to the last statement in the output procedure, the return mechanism terminates the sort or merge, and passes control to the next executable statement after the SORT or MERGE statement.

SORT or MERGE INPUT/OUTPUT PROCEDURE Control

The INPUT or OUTPUT PROCEDURE phrases function in a manner similar to Format 1 of the PERFORM statement (the simple PERFORM). For example, naming a section in an OUTPUT PROCEDURE phrase causes processing of that section during the sort/merge operation to proceed as if that section were named in a PERFORM statement. As with the PERFORM statement, processing of the section ends after processing of its last statement. The last statement in Input and Output Procedures can be the EXIT statement. This is useful for documentation purposes.

RELEASE Statement (Sort Function Only)

The RELEASE statement transfers records from an input/output area to the initial phase of a sort operation. This statement is similar to the WRITE statement.

The RELEASE statement can be specified only within an input procedure associated with a SORT statement. Within an input procedure at least one RELEASE statement must be specified.

When the RELEASE statement is processed, the current contents of record-name are placed in the sort file; that is, made available to the initial phase of the sort operation.

Format

```
RELEASE record-name [ FROM identifier ]
```

Record-name must specify a record associated with the SD entry for file-name-1. Record-name can be qualified.

When the FROM identifier phrase is specified, the RELEASE statement is equivalent to the statement MOVE identifier to record-name followed by the statement RELEASE record-name. Moving takes place according to the rules for the MOVE statement without the CORRESPONDING phrase.

Identifier and record-name must not refer to the same storage area.

After the RELEASE statement is processed, the information in record-name is no longer available unless file-name-1 is specified in a SAME RECORD AREA clause, in which case record-name is still available as a record of the other files named in that clause. When the FROM identifier phrase is specified, the information is still available in identifier.

When control passes from the input procedure, the sort file consists of all those records placed in it by processing of RELEASE statements.

RETURN Statement

The RETURN statement transfers records from the final phase of a sort or merge operation to an input/output area. This statement is similar to the READ statement.

The RETURN statement can be specified only within an output procedure associated with a SORT or MERGE statement. Within an output procedure at least one RETURN statement must be specified.

Format

```
RETURN file-name RECORD [ INTO identifier ] AT END imperative-statement
```

When the RETURN statement is processed, the next record from file-name is made available for processing by the output procedure.

File-name must be described in a Data Division SD entry.

If more than one record description is associated with file-name, these records automatically share the same storage; that is, the area is implicitly redefined. After RETURN statement processing, only the contents of the current record are available; if any data items lie beyond the length of the current record, their contents are undefined.

When the INTO identifier phrase is specified, the RETURN statement is equivalent to the statement RETURN file-name followed by the statement MOVE record-name TO identifier. Moving takes place according to the rules for the MOVE statement without the CORRESPONDING phrase. Any subscripting or indexing associated with identifier is evaluated after the record has been returned and immediately before it is moved to identifier.

The record areas associated with file-name and identifier must not be the same storage area.

After all records have been returned from file-name, the AT END imperative-statement is processed, and no more RETURN statements can be processed.

SORT/MERGE Programming Notes

Sort/merge run time can vary greatly and depends on the following factors:

- The size of the storage pool in which the job runs
- The number of records sorted
- Record size
- The number of key fields specified
- The collating sequence used
- The number and types of input files
- The allocation of USING/GIVING files
- Message severity.

Storage Pool Size: A minimum storage pool size of 200 K is recommended, and 300 K is recommended for the average storage pool size. Processing sort/merge operations and other applications, especially interactive applications, in the same storage pool increases response time and increases sort/merge run time.

Number of Records: Run time increases when the number of records included in the sort increases. The sort/merge operation builds a work record for each input record in the sort. For efficiency, pass only required records to the sort.

Record Size: Run time increases when the records are longer. To minimize run time, do not include fields that contain unnecessary information.

Number of Key Fields: Run time increases when there are more sort/merge key fields. As keys, character fields are more efficient than packed or binary signed fields.

Alternate Collating Sequences: Using an alternate collating sequence increases the run time of the sort. Additional logic is required to change each key field from the standard collating sequence to the alternate collating sequence.

Number and Types of Files: Because the sort must move each record two times (from the file to the work record area, and from the work file to the output file), consider the characteristics of the file.

Allocation of USING/GIVING Files: Failure to open a file required for a `USING` or `GIVING` operation will result in the termination of the program. Users should therefore consider using the Allocate Object (`ALCOBJ`) command, before calling the `COBOL` program in order to ensure that the required files are available.

A logical file on the AS/400 system can be a subset of a physical file, a combination of physical files, and/or a restructuring of a group of fields from one or more physical files. Because of the added flexibility, use of logical files as input to the sort/merge operation can increase the run time of the sort/merge operation. If possible, use a physical file for input to the sort/merge.

Processing a file from a unit record device, such as a diskette, is much slower than processing a file from the data base or from tape. If a file is heavily used during a job or a series of jobs, run time can be improved by copying the file into the data base.

Message Severity: In cases where an input or output procedure transfers control to points outside the procedure, or where control transfers to inside an input or output procedure from elsewhere in the Procedure Division, conditional level message CBL0492 is issued, but compilation does not fail.

In some cases, however, it may be desirable to have compilation fail. If desired, you can force such transfers of control to cause the compilation to fail by changing the conditional level message to a severe level message. This can be done by changing the severity of the message to 30 using the Change Message Description (`CHGMSGD`) `CL` command. See the *CL Reference* for more information about the `CHGMSGD` command. See Appendix D, “COBOL Message Descriptions” for more information about message severities.

SEGMENTATION FEATURE

It is not necessary to be concerned with storage management when writing System/38-Compatible COBOL programs. Segmentation, however, is available for compatibility with other systems.

The segmentation feature provides programmer-controlled storage optimization of the Procedure Division by allowing that division to be subdivided both physically and logically.

Segmentation Concepts

Although it is not required, the Procedure Division of a source program is usually written as a consecutive group of sections, each of which is made up of a series of related operations that process a particular function. Thus, the entire Procedure Division is made up of a number of logical subdivisions. Segmentation allows the programmer to physically divide the Procedure Division into segments, each of which has specific physical and logical attributes.

When Segmentation is used, the entire Procedure Division must be divided into sections. Each section must then be classified as to its physical and logical attributes. Classification is specified by means of segment-numbers. All sections given the same segment-number make up one program segment.

Segment-numbers must be integers from 0 through 99.

Program Segments

There are three types of program segments: fixed permanent, fixed overlayable, and independent.

Fixed Segments

Fixed permanent segments and fixed overlayable segments make up the fixed portion, the part of the Procedure Division that is logically treated as if it were always physically present in main storage. Fixed-portion segment-numbers must be integers from 0 through 49.

A fixed permanent segment is always made available in its last-used state.

A fixed overlayable segment is logically always in main storage during program processing; therefore, it is always available in its last-used state. Any overlay of such a segment is transparent to the user. Thus, a fixed overlayable segment is logically identical with a fixed permanent segment.

Independent Segments

Logically, an independent segment can overlay and be overlaid by other segments during program processing.

An independent segment is made available in its initial state the first time control is passed to it (explicitly or implicitly) during program processing.

An independent segment is made available in its initial state during subsequent transfers of control when:

- The transfer is the result of an implicit transfer of control between consecutive statements that are in different segments (that is, when control drops through into the independent segment from the physically preceding segment).
- The transfer is the result of an implicit transfer from a `SORT` or `MERGE` statement in one segment to a `SORT` input procedure or `SORT/MERGE` output procedure in an independent segment.
- An explicit transfer of control from a section with a different segment-number takes place (as, for example, during the transfer of control in a `PERFORM n TIMES` statement).

An independent segment is made available in its last-used state during subsequent transfers of control when:

- With the exception of the two preceding kinds of implied transfers, an implicit transfer from a section with a different priority takes place (as, for example, when control is returned to the independent segment from a Declarative procedure).
- An explicit transfer results from an `EXIT PROGRAM` statement.

Independent segments must be assigned segment-numbers 50 through 99.

Segmentation Logic

In a segmented program, the sections are classified by a system of segment-numbers according to the following criteria:

- Frequency of Reference–Much used sections, or those that must be available for reference at all times, should usually be within fixed permanent segments. Less frequently used sections should usually be within either fixed overlayable or independent segments, depending on the program logic.
- Frequency of Use–The more frequently a section is referred to, the lower its segment-number; the less frequently it is referred to, the higher its segment-number.
- Logical Relationships–Sections that frequently communicate with each other should be given identical segment-numbers.

Segmentation Control

Except for specific transfers of control, the logical sequence and the physical sequence of program instructions are the same. The compiler inserts any instructions necessary to initialize a segment. It is not necessary to transfer control to the beginning of a segment, or to the beginning of a section within a segment. Instead, control can be transferred to any paragraph in the Procedure Division.

COBOL Source Program Considerations

The following elements of a COBOL source program implement the Segmentation feature:

- The SEGMENT-LIMIT clause in the OBJECT-COMPUTER paragraph of the Environment Division. This clause allows the programmer to control the specification of fixed permanent and fixed overlayable segments.
- Procedure Division segment-numbers, which group sections into segments. The segment numbering scheme also allows specifications of independent segments, fixed permanent segments, and (in conjunction with the SEGMENT-LIMIT clause) of fixed overlayable segments.

Segmentation–Environment Division

In the OBJECT-COMPUTER paragraph, the SEGMENT-LIMIT clause allows the user to reclassify fixed permanent segments while retaining the properties of fixed portion segments for the reclassified segments.

Format

```
[ , SEGMENT-LIMIT IS segment-number ] .
```

The SEGMENT-LIMIT clause allows the programmer to specify certain permanent segments as capable of being overlaid by independent segments without losing the logical properties of fixed portion segments.

Segment-number must be an integer ranging in value from 1 through 49.

When the SEGMENT-LIMIT clause is specified:

- Fixed permanent segments are those with segment-numbers from 0 up to, but not including, the segment-number specified.
- Fixed overlayable segments are those with segment-numbers from the segment-number specified through 49.

For example, if SEGMENT-LIMIT IS 25 is specified, sections with segment-numbers 0 through 24 are fixed permanent segments, and sections with segment-numbers 25 through 49 are fixed overlayable segments.

When the SEGMENT-LIMIT cause is omitted, all sections with segment-numbers 0 through 49 are fixed permanent segments.

Segmentation–Procedure Division

In the Procedure Division of a segmented program, section classification is specified through segment-numbers in the section headers.

Format

```
section-name SECTION [ segment-number ] .
```

All sections with the same segment-number make up one program segment. Such sections need not be contiguous in the source program.

The segment-number must be an integer from 0 through 99.

Segments with segment-numbers 0 through 49 are in the fixed portion of the program. Declarative sections can be assigned only these segment-numbers.

Segments with segment-numbers from 50 through 99 are independent segments.

If the segment-number is omitted from the section header, the segment-number is assumed to be 0.

Segmentation–Special Considerations

When segmentation is used, there are restrictions on the ALTER, PERFORM, SORT, and MERGE statements.

There are also special considerations for calling and called programs.

ALTER Statement

A GO TO statement in an independent segment must not be referred to by an ALTER statement in a different segment. All other uses of the ALTER statement are valid and are processed, even if the GO TO statement referred to is in a fixed overlayable segment.

PERFORM Statement

A PERFORM statement in the fixed portion can have in its range, in addition to any Declarative procedures whose processing is caused within that range, only one of the following:

- Sections and/or paragraphs in the fixed portion
- Sections and/or paragraphs contained within a single independent segment.

A PERFORM statement in an independent segment can have within its range, in addition to any Declarative procedures whose processing is caused within that range, only one of the following:

- Sections and/or paragraphs in the fixed portion
- Sections and/or paragraphs wholly contained in the same independent segment as the PERFORM statement.

SORT and MERGE Statements

If a SORT or MERGE statement appears in the fixed portion, then any SORT input procedures or SORT/MERGE output procedures must appear completely in one of the following:

- The fixed portion
- A single independent segment.

If a SORT or MERGE statement appears in an independent segment, then any SORT input procedures or SORT/MERGE output procedures must appear completely in one of the following:

- The fixed portion
- The same independent segment as the SORT or MERGE statement.

Calling and Called Programs

The CALL statement can appear anywhere within a segmented program. When a CALL statement appears in an independent segment, that segment is in its last-used state when control is returned to the calling program.

Inter-Program Communication Function

Complex data processing problems are often solved by the use of separately compiled but logically interdependent programs which, at run time, form logical and physical subdivisions of a single run unit. A run unit is the total program necessary to solve a data processing problem; it includes one or more programs, and can include programs from source programs written in languages other than COBOL. See “Inter-Program Communication Considerations” in Chapter 7, “System/38-Compatible COBOL Programming Considerations” for more information on COBOL and non-COBOL program communication.

Inter-Program Communication Concepts

When the solution of a problem is subdivided into more than one program, the constituent programs must be able to communicate with each other through transfers of control and/or through reference to common data.

Transfers of Control

In the Procedure Division, a calling program can transfer control to a called program, and a called program can itself transfer control to yet another called program. However, a called program must not directly or indirectly call its caller. For example, if program A calls program B, program B calls program C, and program C then calls program A, the results are unpredictable.

When control is passed to a called program, processing proceeds in the normal way. When a called program processing is completed, the program can either transfer control back to the calling program, call another program, or end the run unit.

Common Data

Program interaction can require that both programs have access to the same data.

In a calling program, the common data items are described in the same manner as other File and Working-Storage Section items. Storage is allocated for these items in the calling program. In a called program, common data items are described in the Linkage Section. Storage is not allocated to them in the called program. Because a calling program can itself be a called program, common data items can be described in the Linkage Section of the calling program. In this case, storage is not allocated for these items in the calling program itself, but rather in the program that called the calling program. For example, program A calls program B which calls program C. Data items in program A can be described in the Linkage Sections of programs B and C, and the one set of data can be made available to all three programs.

When control is transferred from the calling to the called program, the programmer must furnish a list of the common data items in both programs. The sequence of identifiers in both lists determines the match of identifiers between the calling and called programs. A corresponding pair of identifiers in the list names a single set of

data that is available to both programs. While the called program is run, any reference to one of these identifiers is a reference to the corresponding data of the calling program.

COBOL Language Considerations

In the Data Division of the source programs, the programmer defines the common data items to be used by both the calling and called programs. In the calling program, these items can be defined in the File, Working-Storage, or Linkage Sections. In the called program, these items must be defined in the Linkage Section. Common data items need not have the same name and data description, but they must contain the same number of characters.

In the Procedure Division, the list of common data items is established through the USING phrase, which names those data items available to both programs. In the called program, only those items named in the USING list of the called program are available from the data storage of the calling program.

A CALL statement in the calling program transfers control to the first nondeclarative procedural statement in the called program. When the called program has completed its run, control is returned to the calling program by an EXIT PROGRAM statement. The entire run unit can be ended by a STOP RUN statement in either program.

Data Division–Inter-Program Communication

In the Data Division of a called program, the programmer specifies in the Linkage Section those data items that are common with the calling program.

Format

```
    LINKAGE SECTION.
[
| { 77    } data-name/FILLER clause
| { 01-49 }
[
            [ REDEFINES clause ]
            [ BLANK WHEN ZERO clause ]
            [ INDICATOR clause ]
            [ JUSTIFIED clause ]
            [ OCCURS clause ]
            [ PICTURE clause ]
            [ SIGN clause ]
            [ SYNCHRONIZED clause ]
                                    ]
            [ USAGE clause ] .      | . . .
[ 88 condition-name VALUE clause. ] . . .
[ 66 RENAMES clause. ] . . .
```

The Linkage Section has meaning only if this program functions under control of a CALL statement that contains the USING phrase, or a call statement from another language.

The Linkage Section describes data available within the calling program and referred to in both the calling and called programs. Items described in the Linkage Section do not have space allocated for them in the called program. Procedure Division references to these data items are resolved at object time by equating the reference in the called program to the location used in the calling program. For index-names, no such correspondence is established. Index-name references in the calling and called programs always refer to separate indexes. Index-name values can be passed by first moving them to an index data item and passing that index data item.

Items defined in the Linkage Section can be referred to in the Procedure Division only if they are one of the following:

- Operands of a USING phrase in this program
- Data items subordinate to such a USING phrase operand
- Items associated with such a USING operand (such as condition-names or index-names).

Each Linkage Section record-name and noncontiguous data-name must be unique, because neither can be qualified. Descriptions of each clause valid in the Linkage Section are given under “Data Description” in Chapter 9, “Data Division” The following additional considerations apply.

Record Description Entries

Items that have a hierarchical relationship with one another must be grouped into level-01 records according to the rules for formation of record descriptions. Data description clauses can be used to complete the description of the entry. Except for level-88 condition-names, the VALUE clause must not be specified.

Data Item Description Entries

Items that have no hierarchical relationship with each other can be defined as noncontiguous items with level-number 77. The following clauses are required:

- Level-number 77
- Data-name
- PICTURE or USAGE IS INDEX.

Other data description clauses are optional and, when necessary, can complete the description of the item. Except for level-88 condition-names, the VALUE clause must not be specified.

Procedure Division–Inter-Program Communication

In the Procedure Division, control is transferred between a COBOL program and another AS/400 program by means of the CALL statement.

Reference to common data is provided through the USING phrase, which can be specified in the CALL statement and in the Procedure Division header of the called program.

The CANCEL statement releases storage used by a called program.

The EXIT PROGRAM statement allows termination of called program processing. The STOP RUN statement allows termination of the run unit.

CALL Statement

The CALL statement transfers control from one object program to another within the run unit. The calling program must contain a CALL statement at the point where another program is to be called.

Processing of the CALL statement passes control to the first nondeclarative instruction of the called program. Control returns to the calling program at the instruction following the CALL statement.

Called programs themselves can contain CALL statements, but a called program that contains a CALL statement that directly or indirectly calls the calling program gives unpredictable results.

Format

```
CALL { identifier-1 } [ USING data-name-1 [ . data-name-2 ] . . . ]
     { literal-1    }

     [ ON OVERFLOW imperative-statement ]
```

Literal-1 must be nonnumeric and must conform to the rules for formation of a program-name. The first ten characters of the literal are used to make the correspondence between the calling program and the called program. The literal must specify the program-name of the called program.

If literal-1 is specified, the call is classified as a static call because the PROGRAM-ID is determined at compile time.

Identifier-1 must be an alphanumeric data item. Its contents must conform to the rules for formation of a program-name (see “PROGRAM-ID Paragraph” on page 268). The first ten characters of identifier-1 are used to make the correspondence between the calling and called program.

If identifier-1 is specified, the call is classified as a dynamic call because the PROGRAM-ID is resolved at run time each time a call is made. If literal-1 is specified, the PROGRAM-ID is resolved only once.

CALL statement processing passes control to the called program which becomes part of the run unit. If a CALL statement names a program that does not exist in the job’s library list (*LIBL) at run-time, an error message is issued.

A called program is in its initial state the first time it is called within a run unit, and the first time it is called after a CANCEL statement for the called program has been processed.

On all other entries into the called program, it is in its last-used state, and the reinitialization of any items is the responsibility of the user. See Chapter 7, “System/38-Compatible COBOL Programming Considerations” for more information about calling programs.

If there is not enough storage resource available to accommodate the called program in the subsystem, the ON OVERFLOW phrase specifies the action to be taken.

If the ON OVERFLOW phrase is not specified, results are unpredictable.

The user return code is part of the job attributes. You can write a CL program containing the RTVJOBA command to access the return code to control processing in your application. The return code can also be displayed interactively by any of the interfaces that display job attributes. See the *CL Programmer's Guide* for the list of valid return codes and the *CL Reference* for information on the the RTVJOBA command.

USING Phrase

The USING phrase makes data items in a calling program available to a called program. COBOL supports the passing of arguments to other AS/400 programs. The attributes of the data passed depends on the definition requirements of the called program.

The following discussion of the USING phrase assumes that the calling and called programs are written in COBOL.

Format

```
USING data-name-1 [ , data-name-2 ] . . .
```

In a calling program, the USING phrase is valid for the CALL statement; each USING data-name must be defined as a level-01 or level-77 item anywhere in the Data Division. The maximum number of data-names that can be specified is 30.

IBM Extension

In the USING phrase of the calling program CALL statement, the data-names can have level-numbers other than 01 or 77. These data-names can be indexed or subscripted, and can be qualified.

End of IBM Extension

In a called program entered at the beginning of the nondeclaratives portion, the USING phrase is valid in the Procedure Division header; each USING data-name must be defined as a level-01 or level-77 item in the Linkage Section of the called program.

Formats for these individual items show the correct syntax for specifying the USING phrase.

The USING phrase is specified if, and only if, the called program is to operate under control of a CALL statement and that CALL statement itself contains a USING phrase. That is, for each CALL USING statement in a calling program, there must be a corresponding USING phrase specified in the called program.

The order of appearance of USING data-names in both calling and called programs determines the correspondence of single sets of data available to both programs. The correspondence is positional and not by name. Corresponding data-names must contain the same number of characters, although their data descriptions need not be the same. For index-names, no correspondence is established; index-names in calling and called programs always refer to separate indexes.

The data-names specified in a CALL USING statement name data items available to the calling program that can be referred to in the called program. A given data-name can appear more than once.

In a called program, USING data-names must be defined in the Linkage Section. Of the items defined in the Linkage Section, only those named in the USING phrase are available to the program. Within the called program, USING data-names are processed according to their definition within this program.

When the USING phrase is specified, the program runs as if each reference to a USING data-name in the called program Procedure Division is replaced by a reference to the corresponding USING data-name in the calling program.

Examples that illustrate the USING phrase are given in Table 25 on page 513 and in "Inter-Program Communication Function Examples" later in this chapter.

CANCEL Statement

The CANCEL statement releases the storage occupied by a called program.

Format

```
CANCEL { identifier-1 } [ , identifier-2 ] . . .
       { literal-1    } [ , literal-2    ]
```

Each literal or identifier specified in the CANCEL statement must be nonnumeric. The contents must conform to the rules for formation of a program-name (see "PROGRAM-ID Paragraph" on page 268). The first ten characters of the literal or identifier are used to make the correspondence between the calling and called program.

Each literal or identifier specified in the CANCEL statement must be the same as the literal or identifier specified in the associated CALL statement(s).

Subsequent to the processing of a CANCEL statement, the program referred to by the statement ceases to have any logical relationship to the program in which the CANCEL statement appears.

The program named in a CANCEL statement can be entered in its initial state again after the CANCEL statement is processed: the program is entered in its initial state if a CALL statement that names the program is processed by any program in the run unit. A CALL statement is the only means by which a logical relationship to a canceled program can be reestablished.

A called program is canceled either by being directly referred to as the operand of a CANCEL statement or by the termination of the run unit of which the program is a member.

A CANCEL statement only operates on the program specified, and not on any program that may have been called by the canceled program.

No action results from the processing of a CANCEL statement that names a program that has not been called in the run unit, or that names a program that was called but is at present canceled. In either case, control passes to the next statement.

If a CANCEL statement names a program that does not exist in the library list, an error message is issued.

Called programs can contain CANCEL statements. However, a called program must not contain a CANCEL statement that directly or indirectly cancels a calling program. In this case control is passed to the next statement.

A program named in a CANCEL statement must not refer to any program that has been called and has not yet returned control to the calling program. A program can, however, cancel a program that it did not call. For example, if A calls B and B calls C, when A receives control, it can cancel C; or if A calls B and A calls C, when C receives control, it can cancel B.

Table 25. Common Data Items in Inter-Program Communication

Calling Program Description	Called Program Description
WORKING-STORAGE SECTION. 01 PARAM-LIST. 05 PARTCODE PIC A. 05 PARTNO PIC X(4). 05 U-SALES PIC 9(5). . . . PROCEDURE DIVISION. . . . CALL CALLED-PROG USING PARAM-LIST.	LINKAGE SECTION. 01 USING-LIST. 10 PART-ID PIC X(5). 10 SALES PIC 9(5). . . . PROCEDURE DIVISION USING USING-LIST.

Note: In the calling program, the code for parts (PARTCODE) and the part number (PARTNO) are referred to separately. In the called program, the code for parts and the part number are combined into one data item (PART-ID); therefore in the called program, a reference to PART-ID is the only valid reference to them.

EXIT PROGRAM Statement

The EXIT PROGRAM statement specifies the logical end of a called program (subprogram).

Format

```
EXIT PROGRAM.
```

The EXIT statement must be preceded by a paragraph-name, and it must be the only statement in the paragraph.

When control reaches an EXIT PROGRAM statement in a subprogram, control returns to the point in the calling program immediately following the CALL statement. If control reaches an EXIT PROGRAM statement in a main program, control passes

through the exit point to the first sentence of the next paragraph. For more information, see “Inter-Program Communication Considerations” on page 262.

STOP RUN Statement

Format

```
STOP RUN
```

When STOP RUN is specified, processing of the run unit is terminated. If a STOP RUN statement appears in a sequence of imperative statements, it must be the last or the only statement in the sequence. All files should be closed before a STOP RUN statement is processed. If you do not close the files, they are closed by compiler-generated code. For more information, see “Inter-Program Communication Considerations” on page 262.

An implicit STOP RUN is always generated after the last statement in the source program.

Inter-Program Communication Function Examples

A static CALL is illustrated in the following program example.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CALLSTAT.
    .
    .
    .
DATA DIVISION.
    .
    .
    .
WORKING-STORAGE SECTION.
01 RECORD-2  PIC X.
01 RECORD-1.
   05 SALARY PIC S9(5)V99.
   05 RATE      PIC S9V99.
   05 HOURS  PIC S9V99.
    .
    .
    .
PROCEDURE DIVISION
    .
    .
    .
   CALL "PROG" USING RECORD-1, RECORD-2.
    .
    .
    .
   STOP RUN.
```

The following example illustrates a dynamic CALL. The dynamic CALL differs in in the way it is processed from the static CALL that is illustrated in the preceding example.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. CALLDYNA.
  .
  .
  .
DATA DIVISION.
  .
  .
  .
WORKING-STORAGE SECTION.
01 IDENT  PICTURE X(10).
  .
  .
  .
01 RECORD-2  PIC X.
01 RECORD-1.
   05 SALARY  PIC S9(5)V99.
   05 RATE    PIC S9V99.
   05 HOURS   PIC XXX.
  .
  .
  .
PROCEDURE DIVISION.
  .
  .
  .
   MOVE "PROG" TO IDENT.
   CALL IDENT USING RECORD-1, RECORD-2.
  .
  .
  .
   CANCEL IDENT.
  .
  .
  .
   STOP RUN.
```

The following called program can be associated with either of the calling programs in the two preceding examples.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. PROG.
  .
  .
  .
DATA DIVISION.
  .
  .
  .
LINKAGE SECTION.
01 PAYREC.
   10 PAY        PIC S9(5)V99.
   10 HOURLY-RATE   PIC S9V99.
   10 HOURS        PIC S99V9.
01 CODECHAR PIC 9.
  .
  .
  .
PROCEDURE DIVISION USING PAYREC CODECHAR.
  .
  .
  .
EXIT PROGRAM.
```

Processing in these examples begins in the calling program, which can be either CALLSTAT or CALLDYNA. When the first CALL statement is processed, control is transferred to the first statement of the Procedure Division in the called program, PROG.

Note that in each of the calling programs the operand of the first USING phrase is identified as RECORD-1.

When PROG receives control, the values within RECORD-1 are made available to PROG; however, in PROG they are referred to as PAYREC.

Note that the PICTURE character-strings within PAYREC and CODE contain the same number of characters as RECORD-1 and RECORD-2, although the descriptions are not identical.

When processing within PROG reaches the EXIT PROGRAM statement, control is returned to the calling program, and processing continues in that program.

In any given running of these two calling programs, if the values within RECORD-1 are changed between the time of the first CALL and another CALL, the values passed at the time of the second CALL statement are the changed, not the original, values. If the user wishes to use the original values, then he must ensure that they have been saved.

OS/400 Graphics Support

System/38-Compatible COBOL lets you use the CALL statement to access the following OS/400 graphics routines:

- Graphical Data Display Manager (GDDM*), a set of graphics primitives for drawing pictures
- Presentation Graphics Routines (PGR), a set of business charting routines.

You access all these graphics routines with the same format of the CALL statement:

Format

```
CALL "GDDM" USING routine-name [ , data-name-1 ] . . .
```

Routine-name is the name of the graphics routine you want to use.

The data-names that follow routine-name are the parameters necessary to use certain graphics routines. The number of parameters that you must specify varies, depending on which routine you select. When you select a graphics routine, make sure each parameter is the correct size and data type as required by that routine.

The following are examples of calling graphics routines. Remember, you must use the CALL literal format and define each parameter as required by the graphics routine you use.

```
MOVE "FSINIT" TO OS400-GRAPHICS-ROUTINE-NAME.
CALL "GDDM" USING OS400-GRAPHICS-ROUTINE-NAME.
    .
    .
MOVE "GSFLD" TO OS400-GRAPHICS-ROUTINE-NAME.
CALL "GDDM" USING OS400-GRAPHICS-ROUTINE-NAME,
                  PIC-ROW, PIC-COL,
                  PIC-DEPTH, PIC-WIDTH.
```

For more information about graphics routines and their parameters, see the *GDDM Programming Guide*, and the *GDDM Programming Reference*.

DEBUGGING FEATURES

The debugging features specify the conditions under which procedures are to be monitored during the program run.

COBOL source language debugging statements are provided. The user decides what to monitor and what information to retrieve for debugging purposes. The COBOL debugging features simply provide access to pertinent information.

COBOL Source Language Debugging

COBOL language elements that implement the Debugging Feature are a compile-time switch (WITH DEBUGGING MODE), an run-time switch, a USE FOR DEBUGGING Declarative, the special register DEBUG-ITEM, and debugging lines that can be written in the Environment, Data, and Procedure Divisions.

Compile-Time Switch

In the SOURCE-COMPUTER paragraph of the Configuration Section, the WITH DEBUGGING MODE clause acts as a compile-time switch.

Format

```
SOURCE-COMPUTER. computer-name

  [ WITH DEBUGGING MODE ] .
```

The WITH DEBUGGING MODE clause serves as a compile-time switch for the debugging statements written in the source program.

When WITH DEBUGGING MODE is specified, all debugging sections and debugging lines are compiled as specified in this chapter. When WITH DEBUGGING MODE is omitted, all debugging sections and debugging lines are treated as documentation.

Run-Time Switch

The run-time switch dynamically activates the debugging code that is generated when WITH DEBUGGING MODE is specified. Two commands, ENTCBLDBG and ENDCBLDBG are provided to control the run-time switch.

To ensure that the run-time debug switch is set correctly, you should use the System/38-Compatible COBOL CL command ENTCBLDBG within the System/38 environment.

To display the prompt screen, press the F4 key immediately after entering the command. The following screen will be displayed.

```
ENTCBLDBG                    Enter COBOL Debug

Type choices, press Enter.

Program  . . . . . .  _____________   Name
                       *LIBL_____    Name, *LIBL

                                                                  Bottom
F3=Exit   F4=List   F5=Refresh   F11=Keywords   F12=Previous   F13=Prompter help
```

To view the keywords associated with the parameter and option shown above, press the F11 key. The following screen will be displayed.

```
ENTCBLDBG                    Enter COBOL Debug

Type choices, press Enter.

Program  . . . . . . . . . . . . PGM          _________
                                               *LIBL_____

                                                                    Bottom
F3=Exit   F4=List   F5=Refresh   F11=Choices   F12=Previous   F13=Prompter help
```

To set the run-time switch off, enter the command ENDCBLDBG in the System/38 environment.

As indicated above, the prompt screen will be displayed if you press the F4 key.

```
ENDCBLDBG                    End COBOL Debug

Type choices, press Enter.

Program  . . . . . . . . . . . .  _________    Name
                                   *LIBL_____   Name, *LIBL

                                                                    Bottom
F3=Exit   F4=List   F5=Refresh   F11=Keywords   F12=Previous   F13=Prompter help
```

Similarly, By pressing the F11 key the following keyword screen will be displayed.

```
ENDCBLDBG                     End COBOL Debug

Type choices, press Enter.

Program  . . . . . . . . . . . . PGM          _________
                                              *LIBL_____

                                                                    Bottom
F3=Exit   F4=List   F5=Refresh   F11=Choices   F12=Previous   F13=Prompter help
```

The default for the run-time switch is off.

When debugging mode is specified, through the run-time switch, all the debugging sections and debugging lines (D in column 7) compiled into the program are activated.

The ENTCBLDBG command must be entered for each COBOL program (main program or called program) to be debugged in the next COBOL run unit. At the end of the run unit, all run-time switches that are on are set off. If a switch must be set off before the processing of a COBOL run unit, the ENDCBLDBG command should be used.

Run-time switches for up to 15 programs can be on at once.

When the ENTCBLDBG or ENDCBLDBG command is issued in a CL program, concatenation expressions can be used for all parameter values. See the *System/38 CPF Programmer's Guide.* for more information about concatenation expressions.

When debugging mode is suppressed, through the run-time switch, any USE FOR DEBUGGING Declarative procedures are inhibited. However, all debugging lines (D in column 7) remain in effect.

Recompilation of the source program is not required to activate or deactivate the run-time switch.

When WITH DEBUGGING MODE is not specified in the SOURCE-COMPUTER paragraph, the run-time switch has no effect on the running of the program.

USE FOR DEBUGGING Declarative

The USE FOR DEBUGGING sentence in the Procedure Division identifies the items in the source program that are to be monitored by the associated debugging Declarative procedure.

Format

```
                      { [ ALL REFERENCES OF ] identifier-1 }
USE FOR DEBUGGING ON  { file-name-1                        }
                      { procedure-name-1                   }
                      { ALL PROCEDURES                     }

    [ [ ALL REFERENCES OF ] identifier-2 ]
    [ file-name-2                        ] . . .
    [ procedure-name-2                   ]
    [ ALL PROCEDURES                     ]
```

When specified, all debugging sections must be written immediately after the DECLARATIVES header. Except for the USE FOR DEBUGGING sentence there must be no reference to any nondeclarative procedure within the debugging procedure.

Automatic processing of a debugging section is not caused by a statement appearing in a debugging section.

A debugging section for a specific operand is processed only once as the result of the processing of a single statement, no matter how many times the operand is specified in the statement. An exception to this rule is that each specification of a subscripted or indexed identifier where the subscripts or indexes are different causes the calling of the debugging Declarative. For a PERFORM statement that causes repeated processing of a procedure, any associated procedure-name debugging Declarative section is processed each time the procedure is run.

For debugging purposes, each separate occurrence of an imperative verb within an imperative statement begins a separate statement.

Statements appearing outside the debugging sections must not refer to procedure-names defined within the debugging sections.

Except for the USE FOR DEBUGGING sentence itself, statements within a debugging Declarative section can only refer to procedure-names defined in a different USE procedure through the PERFORM statement. Procedure-names within debugging Declarative sections must not appear in USE FOR DEBUGGING sentences.

Table 26 on page 522 defines the points during the program run when the USE FOR DEBUGGING procedures are processed. Identifier-n, file-name-n, and procedure-name-n refer to the first and all subsequent specifications of that type of operand in one USE FOR DEBUGGING sentence. Any particular identifier, file-name, or procedure-name can appear in only one USE FOR DEBUGGING sentence, and only once in that sentence.

An identifier in a USE FOR DEBUGGING sentence:

- Must be specified without the subscripting or indexing normally required if it contains an OCCURS clause or is subordinate to an entry containing an OCCURS clause.

- Must not be a special register.

When ALL PROCEDURES is specified in a USE FOR DEBUGGING sentence, procedure-name-1, procedure-name-2, procedure-name-3, and so on, must not be specified in any USE FOR DEBUGGING sentence. The ALL PROCEDURES phrase can be specified only once in a program.

References to the DEBUG-ITEM special register can be made only from within a debugging Declarative procedure.

Table 26. Processing of Debugging Declaratives

USE FOR DEBUGGING Operand	Upon processing of the following, the USE FOR DEBUGGING procedures are run immediately
identifier-n	Before REWRITE/WRITE identifier-n and after FROM phrase move, if applicable. After each initialization, modification, or evaluation of identifier-n in PERFORM/VARYING/AFTER/UNTIL identifier-n. After any other COBOL statement that explicitly refers to identifier-n and could change its contents. (See note.)
ALL REFERENCES OF identifier-n	Before GO TO DEPENDING ON identifier-n, control is transferred, and before any associated debugging section for procedure-name is processed. Before REWRITE/WRITE identifier-n and FROM phrase move, if applicable. After each initialization, modification or evaluation of identifier-n in PERFORM/VARYING/AFTER/UNTIL identifier-n. After any other COBOL statement explicitly referring to identifier-n. (See note.)
file-name-n	After CLOSE/DELETE/OPEN/START file-name-n. After READ file-name-n where AT END/INVALID KEY was not processed.
procedure-name-n	Before each time the named procedure is run. After processing an ALTER statement referring to the named procedure.
ALL PROCEDURES	Before each time any non-debugging procedure is run. After processing every ALTER statement (except ALTER statements in Declarative procedures).

Notes:

1. Operands acted upon but not explicitly named in such statements as ADD, MOVE, or SUBTRACT CORRESPONDING never cause activation of a USE FOR DEBUGGING procedure when such statements are processed.
2. If an operand is specified in a phrase that is not processed, the associated debugging section is not processed.
3. A SEARCH or SEARCH ALL statement that refers to an identifier that normally requires subscripting or indexing does not call the USE FOR DEBUGGING procedures.
4. When a USE FOR DEBUGGING operand is used as a qualifier, such a reference in the program does not activate the debugging procedures.

DEBUG-ITEM Special Register

The DEBUG-ITEM special register provides information for a debugging Declarative procedure. DEBUG-ITEM has the following implicit description.

```
01 DEBUG-ITEM.
   02 DEBUG-LINE      PICTURE IS X(6).
   02 FILLER          PICTURE IS X VALUE SPACE.
   02 DEBUG-NAME      PICTURE IS X(30).
   02 FILLER          PICTURE IS X VALUE SPACE.
   02 DEBUG-SUB-1     PICTURE IS S9999 SIGN IS
                      LEADING SEPARATE CHARACTER.
   02 FILLER          PICTURE IS X VALUE SPACE.
   02 DEBUG-SUB-2     PICTURE IS S9999 SIGN IS
                      LEADING SEPARATE CHARACTER.
   02 FILLER          PICTURE IS X VALUE SPACE.
   02 DEBUG-SUB-3     PICTURE IS S9999 SIGN IS
                      LEADING SEPARATE CHARACTER.
   02 FILLER          PICTURE IS X VALUE SPACE.
   02 DEBUG-CONTENTS  PICTURE IS X(n).
```

The DEBUG-ITEM special register provides information about the conditions causing debugging section processing.

Before each debugging section is processed, DEBUG-ITEM is filled with spaces. The contents of the DEBUG-ITEM subfields are then updated according to the rules for the MOVE statement, with one exception: DEBUG-CONTENTS is updated as if the move were an alphanumeric to alphanumeric elementary move without conversion of data from one form of internal representation to another. After updating, each field contains:

- DEBUG-LINE: The compiler-generated statement number, right justified and padded on the left with zeros. For example, 000112.
- DEBUG-NAME: The first 30 characters of the name causing debugging section processing. All qualifiers are separated by the word OF (subscripts or indexes are not entered in DEBUG-NAME).
- DEBUG-SUB-1, DEBUG-SUB-2, DEBUG-SUB-3: If the DEBUG-NAME is subscripted or indexed, the occurrence number of each level is entered in the respective DEBUG-SUB-n. If the item is not subscripted or indexed, these fields remain spaces.

- DEBUG-CONTENTS: Data is moved into DEBUG-CONTENTS as shown in Table 27 on page 524. DEBUG-CONTENTS is the same size as the largest identifier in the program.

Table 27. DEBUG-ITEM Subfield Contents

Item Causing Debug Section Processing	DEBUG-LINE Contains Number of COBOL Statement Referring to	DEBUG-NAME Contains	DEBUG-CONTENTS Contains
identifier-n	identifier-n	identifier-n	Contents of identifier-n when control passes to debug section.
file-name-n	file-name-n	file-name-n	For READ: contents of record retrieved. Other references: spaces.
procedure-name-n ALTER reference	ALTER statement	procedure-name-n	procedure-name-n in TO PROCEED TO phrase
GO TO procedure-name-n	GO TO statement	procedure-name-n	
procedure-name-n in SORT/MERGE INPUT/OUTPUT PROCEDURE	SORT/MERGE statement	procedure-name-n	"SORT INPUT" "SORT OUTPUT" "MERGE OUTPUT" as applicable
PERFORM statement transfer of control	This PERFORM statement	procedure-name-n	"PERFORM LOOP"
procedure-name-n in a USE procedure	Statement causing USE procedure to run	procedure-name-n	"USE PROCEDURE"
Implicit transfer from previous sequential procedure	Previous statement processed in previous sequential procedure (see note)	procedure-name-n	"FALL THROUGH"
First processing of first nondeclarative procedure	Line number of first statement in the procedure	First nondeclarative procedure-name	"START PROGRAM"

Note: If this paragraph is preceded by a section header and control is passed through the section header, the statement number refers to the section header.

Debugging Lines

A debugging line is any line in a source program with a D coded in column 7 (the continuation area). If a debugging line contains nothing but spaces in Area A and Area B, it is considered a blank line.

Each debugging line must be written so that a syntactically correct program results whether the debugging lines are compiled into the program or syntax-checked, but are treated as documentation.

Successive debugging lines are permitted. Debugging lines can be continued. However, each continuation line must contain a D in column 7, and character-strings must not be broken across two lines.

Debugging lines can be specified only after the OBJECT-COMPUTER paragraph.

When the WITH DEBUGGING MODE clause is specified in the SOURCE-COMPUTER paragraph, all debugging lines are compiled as part of the object program.

When the WITH DEBUGGING MODE clause is omitted, all debugging lines are syntax-checked, but are treated as documentation.

FIPS FLAGGER

The FIPS (Federal Information Processing Standard) Flagger can be specified. Depending on the compiler option specified, it identifies source statements and clauses that do not conform to a specified level of the federal standard. For information on the FIPS Flagger, see “Federal Information Processing Standard Flagger” on page 2.

Appendix A. Summary of IBM Extensions

This appendix contains a brief summary of the extensions to System/38 environment COBOL with references to discussions of the extensions elsewhere in this manual.

Character-String Considerations

The maximum length of a nonnumeric literal is 160 characters. See “Literals” on page 7.

Identification Division

The system uses the first 10 characters of the program-name specified in the PROGRAM-ID paragraph. See “PROGRAM-ID Paragraph” in Chapter 8, “Identification and Environment Divisions.”

Environment Division

The DEV parameter of an Override command can change the device type that the file will use. See “FILE-CONTROL Paragraph” on page 281.

FORMATFILE must be specified in the ASSIGN clause to use an externally described printer file. See See “FILE-CONTROL Paragraph” on page 281.

COBOL programs can process data base files. The ORGANIZATION of the file indicates the current program usage. See “File Processing Summary” and “FILE-CONTROL Paragraph, ORGANIZATION Clause” in Chapter 8, “Identification and Environment Divisions.”

The OVRDBF command can set the current record pointer when the file is opened. See “FILE-CONTROL Paragraph, ACCESS MODE Clause” in Chapter 8, “Identification and Environment Divisions.”

The RECORD KEY data item, data-name-2, can be numeric. EXTERNALLY-DESCRIBED-KEY can be specified in the RECORD KEY clause. See “FILE-CONTROL Paragraph, RECORD KEY Clause (Indexed File)” in Chapter 8, “Identification and Environment Divisions.”

The DUPLICATES phrase can be specified for the RECORD KEY clause. See “FILE-CONTROL Paragraph, RECORD KEY Clause (Indexed File)” in Chapter 8, “Identification and Environment Divisions.” Additional information is given in discussions of the READ, REWRITE, DELETE, and WRITE statements in Chapter 10, “Procedure Division.”

The keywords specified for the data item in DDS can modify record sequence. In particular if the DDS keyword DESCEND is used when the field is specified as a key, the sequence can be a descending key sequence. See “FILE-CONTROL Paragraph, RECORD KEY Clause (Indexed File)” in Chapter 8, “Identification and Environment Divisions.”

The COMMITMENT CONTROL clause can be specified to enable the synchronizing or canceling of data base changes, and to provide additional record locking for records being changed. See “I-O-CONTROL Paragraph, COMMITMENT CONTROL

Clause" in Chapter 8, "Identification and Environment Divisions" and "Commitment Control Considerations" on page 247.

Data Division

Elementary items or group items immediately subordinate to one group item can have unequal level-numbers. See "Data Description Concepts, Level-Numbers" in Chapter 9, "Data Division."

The OVRTAPF command can change the LABEL RECORDS clause at run time. See "File Description Entry, LABEL RECORDS Clause" in Chapter 9, "Data Division."

If the CODE-SET clause is omitted, the CODE parameter of the CRTDKTF or the CRTTAPF command is used. The OVRDKTF or the OVRTAPF command can change the CODE-SET clause at run time. See "File Description Entry, CODE-SET Clause" in Chapter 9, "Data Division."

For the USING phrase of the CALL statement, data-names can have level-numbers that are not 01 or 77, and the data-names can be indexed, subscripted, or qualified. See "CALL Statement, USING Phrase" in Chapter 11, "Using the Additional COBOL Functions."

A FILLER item can be used as a group item definition. See "Data-Name or FILLER Clause" in Chapter 9, "Data Division."

COMPUTATIONAL-3 (packed decimal) and COMPUTATIONAL-4 (binary) can be specified for the USAGE clause of numeric items. See "USAGE Clause" on page 322.

The key specified for an OCCURS clause can have USAGE of COMPUTATIONAL-3 or COMPUTATIONAL-4. See "OCCURS Clause" on page 476.

The JUSTIFIED clause can be specified for alphanumeric edited items. See "JUSTIFIED Clause" in Chapter 9, "Data Division."

Procedure Division

The mnemonic-names OPEN-FEEDBACK and I-O-FEEDBACK are used for file information and are accessed through an ACCEPT statement format. See "ACCEPT Statement" on page 373.

THEN is used as a separator on the IF statement. See "IF Statement" in Chapter 10, "Procedure Division."

The AT END phrase can be omitted for the READ statement. See "Common Input/Output Phrases" on page 370.

The FORMAT phrase is valid for DELETE, READ, REWRITE, START, and WRITE statements. See "DELETE Statement", "READ Statement", "REWRITE Statement", "START Statement", and "WRITE Statement" in Chapter 10, "Procedure Division."

The special register DB-FORMAT-NAME contains information about the processing of file input/output statements. See "Common Input/Output Phrases, DB-FORMAT-NAME Special Register" in Chapter 10, "Procedure Division."

The identifier in an ACCEPT statement can have USAGE of COMPUTATIONAL-3 or COMPUTATIONAL-4. See "ACCEPT Statement" on page 373.

The ACCEPT statement can be used to transfer data from a job's local data area to a specified data item. See "ACCEPT Statement" on page 373.

The system always rewinds and unloads the tape when REEL/UNIT is specified in the CLOSE statement. See "CLOSE Statement" on page 377.

The INHWRT parameter of the OVRDBF command can inhibit the DELETE, READ, and WRITE statements. See "DELETE Statement", "READ Statement", and "WRITE Statement" in Chapter 10, "Procedure Division."

For a file with duplicate primary keys allowed, a READ statement must immediately precede a DELETE or REWRITE statement to ensure proper deletion. See "DELETE Statement" and "REWRITE Statement" in Chapter 10, "Procedure Division."

For the DISPLAY statement, COMPUTATIONAL-4 items are converted to zoned decimal items and signed noninteger numeric literals are allowed. See "DISPLAY Statement" on page 385.

The DISPLAY statement can be used to transfer data to a job's local data area. See "DISPLAY Statement" on page 385.

A logical file opened for OUTPUT does not remove all records in the physical file on which it is based. The OVRDBF command can specify the first record to be made available to the program at run time. See "OPEN Statement" in Chapter 10, "Procedure Division."

FIRST, PRIOR, and LAST can be specified on the READ statement for indexed files with dynamic access. See "READ Statement" in Chapter 10, "Procedure Division."

The KEY phrase of the START statement can specify EXTERNALLY-DESCRIBED-KEY. A comparison can be affected by the type of key fields in the record area defined for the file. See "START Statement" in Chapter 10, "Procedure Division."

For the WRITE statement, the mnemonic-name phrase can be used for stacker selection on a card punch file. See "WRITE Statement"

Note: Card devices are not supported by System/38-Compatible COBOL, even though the devices are accepted by the syntax checker. in Chapter 10, "Procedure Division."

The composite of all operands in an arithmetic statement has a maximum of 30 digits. See "Arithmetic Statement Operands" in Chapter 10, "Procedure Division."

In the CORRESPONDING phrase of an arithmetic statement, the identifiers d1 and d2 can be subordinate to a FILLER item. See "Common Phrases, CORRESPONDING Phrase" in Chapter 10, "Procedure Division."

The two additional formats of the SET statement can be used to set mnemonic-names to on or off and to set condition-names to true. See "SET Statement" in Chapter 10, "Procedure Division."

Two active PERFORM and GO TO statements can have a common exit point. See "PERFORM Statement" in Chapter 10, "Procedure Division."

Input files do not need to be sequenced before a merge operation. See "SORT/MERGE" on page 491.

In SORT/MERGE, input or output procedures can transfer control outside the input or output procedure using a PERFORM statement, or have control transferred inside the input or output procedure from elsewhere in the Procedure Division using a PERFORM statement, without causing compilation to fail. See “SORT/MERGE” on page 491.

The COMMIT statement can be used to synchronize changes to records in data base files under commitment control, while preventing other jobs from accessing or modifying those records until the COMMIT is complete. See “COMMIT Statement” in Chapter 10, “Procedure Division.”

The ROLLBACK statement can be used to cancel data base changes from files under commitment control when the changes should not remain permanent. See “ROLLBACK Statement” in Chapter 10, “Procedure Division.”

COPY Statement–All Divisions

Pseudo-text for the REPLACING phrase of the COPY statement has considerations for division, section, and paragraph entries, and also for the placement of copied text as it appears in pseudo-text-2. See “COPY Statement, Replacing Phrase” in Chapter 11, “Using the Additional COBOL Functions.”

The file-name is optional. The default file-name is QCBLSRC. See “Qualification Rules” on page 17.

The Format 2 COPY statement is used to create Data Division entries for externally described files in a program. See “COPY Statement” on page 30.

TRANSACTION Files

The data organization for work stations, display files, BSC files, communications files, and mixed files is TRANSACTION. TRANSACTION files have special formats for the file-control entry, file description entry, and the input/output statements. TRANSACTION file considerations are in Chapter 5, “Interactive Processing Considerations and Example Programs.”

Considerations for the TRANSACTION file-control entry include those for:

- The ASSIGN clause
- The ORGANIZATION clause
- The ACCESS MODE clause
- The FILE STATUS clause
- The CONTROL-AREA clause.

Considerations for the TRANSACTION file description entry are the same as those for other file description entries.

Boolean data provides a means of modifying and passing the values of the indicators associated with the display screen formats. See “Indicators” on page 92.

The ACCEPT statement provides a way of accessing information about a program device when function-name is associated with a mnemonic-name of ATTRIBUTE-DATA in the SPECIAL-NAMES paragraph. See “ACCEPT Statement.”

TRANSACTION file considerations for OPEN, CLOSE, READ, WRITE, REWRITE, and USE statements are given under the discussions for the respective statements and the discussions of the FORMAT, TERMINAL, INDICATORS, NO DATA, and SUBFILE phrases in Chapter 5, “Interactive Processing Considerations and Example Programs.”

The ACQUIRE statement can be used to acquire a program device for a TRANSACTION file. See “ACQUIRE Statement” in Chapter 5, “Interactive Processing Considerations and Example Programs.”

The DROP statement can be used to release a program device acquired by a TRANSACTION file. See “DROP Statement” in Chapter 5, “Interactive Processing Considerations and Example Programs.”

Compiler Options

Sequence checking can be suppressed at compile time. The apostrophe or the quotation mark can be used as a separator according to the compiler option specified. See “Create COBOL Program Command” on page 37 and “PROCESS Statement” on page 46.

Appendix B. Associated Card File Processing

Card files (such as READER, PUNCH, and PUNCHPRINT), card devices, and related language elements are not supported by System/38-Compatible COBOL in the System/38 environment. Programs written with references to card devices and related language elements will be syntax checked but compilation in the System/38 environment will fail. These programs are compatible with the System/38 and compilation of such programs can be performed on a System/38 on which the COBOL licensed program (Program 5714-CB1) is installed.

The 5424 Multi-Function Card Unit (MFCU) can process more than one card processing function in a single pass through the unit. If a card has already been partially punched, the MFCU can read the card, punch additional information into the card, and print up to 128 characters of information on the card. COBOL supports these combined functions through normal control language. This support is based on the concept of associated files.

COBOL handles each combined function as a separate logical file; each such logical file has its own file structure and processing requirements. Therefore, the user must define each function as if it were a unique file. However, because such combined function files refer to one physical unit, the user must define the logical files as being associated with each other, and relate them to each other during processing. The following sections explain the programming requirements for associated card file processing in System/38-Compatible COBOL.

Environment Division

Associated card file processing requires certain information in the SELECT and ASSIGN clauses.

SELECT Clause

A unique system file-name must be defined for each of the functions (reading, punching, and printing) to be combined.

ASSIGN Clause

For associated card files, the ASSIGN clause assignment-name must specify the primary (P) hopper of the 5424 MFCU and the association. The following format is valid:

Format

```
{ READER     }
{ PUNCH      } [ - system-name ] - P-association
{ PRINT      }
{ PUNCHPRINT }
```

The association must be the same one-digit integer for all associated logical files. This tells the compiler that each logical file is part of a particular associated card file processing structure that is assigned to one physical unit. Any two, or all three, of the functions READER, PUNCH, or PRINT can each be specified once. When the

function PUNCHPRINT is specified for an associated file, READER can be the only other function specified for that association.

More than one associated card file processing structure can be defined in a program; however, the structures must not be processed concurrently. Each such structure must have a unique association entry.

Data Division

An FD entry and a 01 record description entry must be defined in the File Section of the Data Division for each associated logical file.

Procedure Division

All associated files within one associated card file processing structure must be opened before a READ or WRITE statement is processed for any file in the structure. Similarly, no associated files can be closed until all READ and WRITE statements have been processed. When all such statements have been processed, all of the associated files must be closed.

An OPEN, READ, WRITE, or CLOSE statement for one of the associated files cannot be processed without concurrent processing of the other associated files. That is, all processing of associated files must reflect the association.

For associated files with the functions of read (READER), punch (PUNCH), print (PRINT), and punchprint (PUNCHPRINT), the following processing rules apply:

1. An OPEN statement must be processed for each file; with the INPUT phrase specified for the READER file and the OUTPUT phrase specified for the PUNCH, PRINT, and PUNCHPRINT files.
2. To make the logical input record available, a READ statement must first be processed for the READER file.
3. The next input/output operation processed for an associated file must be a WRITE statement for the associated PUNCH file. However, before the WRITE statement is processed, all data to be punched must be moved to the record specified for the associated PUNCH file.

   Note: An alternative method is to move the additional data to be punched to the appropriate fields of the input record, and then to process a WRITE statement for the PUNCH file using the FROM identifier phrase, where the identifier is the record-name specified for the READER file.
4. The WRITE statement for the PRINT file must be the next input/output statement processed for any of the associated files. Before the WRITE statement for the PRINT file is processed, you must format the data to be printed by moving it to the record specified for the PRINT file.
5. Steps 1 through 4 are repeated for all other records in the READER file.
6. When the PUNCHPRINT file is specified, a WRITE statement for the PUNCHPRINT file must be the next input/output statement following the READ statement for the READER file. Before the WRITE statement for the PUNCHPRINT file is processed, you must format the data to be punched and printed by moving it to the record specified for the PUNCHPRINT file.

When both PUNCH and PRINT files are associated, the stacker selection, if specified, in the WRITE statement for the PRINT file overrides any stacker selection specified for the PUNCH file.

When the READER file is specified for an associated card file, a READ until the AT END condition is detected must be done. This forces punch and/or print operations to occur for the last record read.

The order of processing specified in the preceding paragraphs also applies to a structure where only two of the three functions are associated, except that the processing of the unspecified function is not required.

Appendix C. Intermediate Result Fields

This appendix discusses the conceptual compiler algorithms for determining the number of integer and decimal places reserved for intermediate results. The following abbreviations are used:

i	Number of integer places carried for an intermediate result.
d	Number of decimal places carried for an intermediate result.
dmax	In a particular statement the larger of either: • The number of decimal places needed for the final result field(s) • The maximum number of decimal places defined for any operand except exponents and divisors.
op1	First operand in a generated arithmetic statement.
op2	Second operand in a generated arithmetic statement.
d1,d2	Number of decimal places defined for op1 or op2, respectively.
ir	Intermediate result field obtained from the processing of a generated arithmetic statement or operation. Intermediate results are represented by ir1, ir2, and so on. Successive intermediate results may share the same memory location.

When an arithmetic statement contains only a single pair of operands, no intermediate results are generated. Intermediate results are possible in the following cases:

- In an `ADD` or `SUBTRACT` statement containing multiple operands immediately following the verb
- In a `COMPUTE` statement specifying a series of arithmetic operations
- In arithmetic expressions contained in an `IF` or `PERFORM` statement
- In the `GIVING` option with multiple result fields for the `ADD`, `SUBTRACT`, `MULTIPLY`, `DIVIDE`, or `COMPUTE` statements.

In such cases, the compiler treats the statement as a succession of operations. For example, the following statement:

```
COMPUTE Y = A + B * C - D / E + F ** G
```

is replaced by

```
F**G                                    yielding ir1
MULTIPLY B          BY C                yielding ir2
DIVIDE E            INTO D              yielding ir3
ADD A               TO ir2              yielding ir4
SUBTRACT ir3        FROM ir4            yielding ir5
ADD ir5             TO ir1              yielding Y
```

Compiler Calculation of Intermediate Results

The number of integer places in an ir is calculated as described in the following paragraphs:

The compiler first determines the maximum value that the ir can contain by processing the statement in which the ir occurs.

- If an operand in this statement is a data-name, the value used for the data-name is equal to the numerical value of the `PICTURE` for the data-name (that is, `PICTURE 9V99` has the value 9.99).
- If an operand is a literal, the literal is treated as though it had a `PICTURE`, and the numerical value of the `PICTURE` is used (that is, the literal +127.3 has an implied `PICTURE S999V9`).
- If an operand is an intermediate result, the `PICTURE` determined for the intermediate result in a previous operation is used. The numerical value of that `PICTURE` is used.
- If the operation is division:
  - If op2 is a data-name, the value used for op2 is the minimum nonzero value of the digit in the `PICTURE` for the data-name (that is, `PICTURE` 9V99 has the value 0.01).
  - If op2 is an intermediate result, the intermediate result is treated as though it had a `PICTURE`, and the minimum nonzero value of the digits in this `PICTURE` is used.

When the maximum value of the ir is determined by the above procedures, i is set equal to the number of integers in the maximum value.

The number of decimal places contained in an ir is calculated as:

Operation	Decimal Places
+ or –	d1 or d2, whichever is greater
*	d1 + d2
/	d1 – d2 or dmax, whichever is greater
**	dmax if op2 is nonintegral or a data-name; d1 * op2 if op2 is an integral literal

Note: The user must define the operands of any arithmetic statement with enough decimal places to give the desired accuracy in the final result.

Table 28 indicates the action of the compiler when handling intermediate results.

Table 28. Compiler Action on Intermediate Results

Value of i = d	Value of d	Value of i + dmax	Action Taken
<30 = 30	Any value	Any value	*i* integer and *d* decimal places are carried for ir
>30	<dmax =dmax	Any value	30 - *d* integer and *d* decimal places are carried for ir
	>dmax	<30 = 30	*i* integer and 30 - *i* decimal places are carried for ir
		>30	30 - dmax integer and dmax decimal places are carried for ir

Appendix D. COBOL Message Descriptions

This appendix contains a general introduction to System/38-Compatible COBOL messages supplied by IBM with the COBOL/400 licensed program.

Interactive Messages

In an interactive environment, messages are displayed on the work station screen. They can appear on the current display as a result of processing a program or in response to your keyed input to prompts, menus, or command entry displays. The messages can also appear on request, as a result of a display command or an option on a menu.

The interactive messages for the COBOL licensed program begin with a CSC, a CBE, or a CBL prefix.

The CSC messages are issued by the COBOL syntax checker when SEU is used to enter your COBOL source.
The CBE messages provide you with additional information about system operation during run-time.
The CBL messages are compiler-generated messages. See Compilation Messages below for a further description.

Message numbers are assigned as follows:

Error Message	Description
CBE7000 through CBE7199	Escape Messages
CBE7200 through CBE7999	Run-time messages
CBE9001	Escape message
CBL0000 through CBL 0999	Messages with severity less than 30
CBL1000 through CBL1999	Messages with severity greater than 29
CBL8000 through CBL8999	FIPS Flagger messages
CSC0000 through CSC1999	Syntax checker messages

Compilation Messages

Compiler-generated messages indicate conditions encountered during program compilation.

The CBL messages are printed in the program listing and include the messages issued when FIPS (Federal Information Processing Standard) flagging is requested.

See Figure 16 on page 57 for an example of COBOL Messages which follow a source listing.

Severity Levels

System/38-Compatible COBOL provides the following message severity levels:

Severity	Meaning
00	Informational: This level is used to convey information to the user that may be of interest to him. No error has occurred. Informational messages are listed only when the FLAG (00) option is specified.
10	Warning: This level indicates that an error was detected but is not serious enough to interfere with the running of the program.

20 Conditional: This level indicates that an error was made, but the compiler is taking a recovery that might yield the desired code.

30 Error: This level indicates that a serious error was detected. Compilation is completed, but running of the program cannot be attempted.

40 Unrecoverable: This level usually indicates a user error that forces termination of processing.

50 Unrecoverable: This level usually indicates a compiler error that forces termination of processing.

99 Action: Some manual action is required, such as entering a reply, changing printer forms, or replacing diskettes.

Note: 00, 10, and 20 messages are suppressed when the FLAG(30) option of the PROCESS statement is used or the CRTCBLPGM command specifies FLAG(30) and is not overridden by the PROCESS statement. See "PROCESS Statement" on page 46 for further information.

The compiler always attempts to provide full diagnostics of all source text in the program, even when errors have been detected. If the compiler cannot continue on a given statement, the message states that the compiler cannot continue and that it will ignore the rest of the statement. When this occurs, the programmer should examine the entire statement.

The OS/400 message facility is used to produce all messages. The System/38-Compatible COBOL compiler messages reside in the message file QCBLMSG, and the run-time messages reside in the message file QCBLMSGE.

Substitution variables and valid reply values are determined by the program sending the message, *not* by the message description stored in the message file. However, certain elements of a message description can be changed: for example, the text, severity level, default response, or dump list. To effect such changes, you need to define another message description using an Add Message Description (ADDMSGD) command, place the modified description in a user-created message file,[17] and specify that file in the Override Message File (OVRMSGF) command. Using the OVRMSGF command allows the compiler to retrieve messages from the specified file. See the ADDMSGD and OVRMSGF commands in the *CL Programmer's Guide*, the *CL Reference*, and in the *System/38 Environment Programmer's Guide/Reference* for additional information.

CAUTION: Overriding an IBM-supplied message with a user-created message can produce results you do not anticipate. If reply values are not retained, the program might not respond to any replies. Changing default replies on *NOTIFY type messages could affect the ability of the program to run in unattended mode. Changing the severity could cancel a job not previously canceled. Be cautious when overriding IBM-supplied messages with user-created messages.

[17] If an IBM-supplied message must be changed and replaced in *its* message file, call your service representative.

Appendix E. File Structure Support Summary and Status Key Values

Table 29 lists the required and optional entries for various types of file structures supported. Any file with a device type of disk can be assigned to a data base or non-data base auxiliary storage file. The codes used are as follows:

.	Not applicable
B	Optional for a work station that supports subfiles
C	Optional entry, treated as comments only
D	Optional for file assigned to -DATABASE, not allowed if not assigned to a data base file
I	Optional for a file opened for input or input-output
O	Optional
R	Required
S	Required for a work station that supports subfiles
X	Required; syntax-checked, but treated as documentation.

Table 30 on page 548 contains status key values and their meanings.

Table 29 (Page 1 of 4). File Structure Support

DEVICE TYPE	CARD READER	CARD PUNCH	CARD PRINT	CARD PUNCHPRINT	PRINTER	TAPE	DISK SEQ	DISK REL SEQ	DISK REL RANDOM	DISK REL DYNAMIC	DISK IDX SEQ	DISK IDX RANDOM	DISK IDX DYNAMIC	WORK STATION	DISKETTE	FORMATFILE
Environment Division																
RERUN...RECORDS	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
SAME	O	O	O	O	O	O	O	.	O	O	O	O	O	O	O	O
AREA	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
RECORD AREA	O	O	O	O	O	O	O	.	O	O	O	O	O	O	O	O
SORT AREA	C	.	.	.	.	C	C	.	.	.	.	.	.	.	.	.
SORT MERGE AREA	C	.	.	.	.	C	C	.	.	.	.	.	.	.	.	.
MULTIPLE FILE TAPE	.	.	.	.	.	C	.	.	.	.	.	.	.	.	.	.
COMMITMENT CONTROL	.	.	.	.	.	.	D	D	D	D	D	D	D	.	.	.
SELECT	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
ASSIGN	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
OPTIONAL	I	.	.	.	.	I	I	.	.	.	.	.	.	.	I	I
ORGANIZATION	O	O	O	O	O	O	O	R	R	R	R	R	R	R	O	O
SEQUENTIAL	O	O	O	O	O	O	O	.	.	.	.	.	.	.	O	O
RELATIVE	.	.	.	.	.	.	.	R	R	R	.	.	.	.	.	.
INDEXED	.	.	.	.	.	.	.	.	.	.	R	R	R	.	.	.
TRANSACTION	.	.	.	.	.	.	.	.	.	.	.	.	.	R	.	.
ACCESS	O	O	O	O	O	O	O	O	R	R	O	R	R	O	O	O
SEQUENTIAL	O	O	O	O	O	O	O	O	.	.	O	.	.	O	O	O
RANDOM	.	.	.	.	.	.	.	.	R	.	.	R	.	.	.	.
DYNAMIC	.	.	.	.	.	.	.	.	.	R	.	.	R	S	.	.
RESERVE	C	C	C	C	C	C	C	C	C	C	C	C	C	.	C	C
RELATIVE KEY	.	.	.	.	.	.	.	O	R	R	.	.	.	S	.	.
RECORD KEY	.	.	.	.	.	.	.	.	.	.	R	R	R	.	.	.
DUPLICATES	.	.	.	.	.	.	.	.	.	.	D	D	D	.	.	.
FILE STATUS	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
CONTROL-AREA	.	.	.	.	.	.	.	.	.	.	.	.	.	O	.	.
DATA DIVISION																
LABEL RECORDS	X	X	X	X	X	R	X	X	X	X	X	X	X	X	X	X
STANDARD	.	.	.	.	.	O	R	R	R	R	R	R	R	O	R	R
OMITTED	R	R	R	R	R	O	.	.	.	.	.	.	.	O	.	.

Table 29 (Page 2 of 4). File Structure Support

DEVICE TYPE	CARD READER	CARD PUNCH	CARD PRINT	CARD PUNCHPRINT	PRINTER	TAPE	DISK SEQ	DISK REL SEQ	DISK REL RANDOM	DISK REL DYNAMIC	DISK IDX SEQ	DISK IDX RANDOM	DISK IDX DYNAMIC	WORK STATION	DISKETTE	FORMATFILE
VALUE OF	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C	C
BLOCK CONTAINS	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
RECORD CONTAINS	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
DATA RECORDS	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
CODE-SET	.	.	.	.	.	O	.	.	.	.	.	.	.	.	O	.
LINEAGE	.	.	.	.	O	.	.	.	.	.	.	.	.	.	.	.
PROCEDURE DIVISION																
OPEN	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
INPUT	R	.	.	.	.	O	O	O	O	O	O	O	O	.	O	O
OUTPUT	.	R	R	R	R	O	O	O	O	O	O	O	O	.	O	O
I-O	.	.	.	.	.	.	O	O	O	O	O	O	O	R	.	O
NO REWIND	.	.	.	.	.	I	.	.	.	.	.	.	.	.	.	.
REVERSED	.	.	.	.	.	I	.	.	.	.	.	.	.	.	.	.
EXTEND	.	.	.	.	.	O	O	.	.	.	.	.	.	.	.	O
CLOSE	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R	R
REEL/UNIT	.	.	.	.	.	O	.	.	.	.	.	.	.	.	.	.
REMOVAL	.	.	.	.	.	O	.	.	.	.	.	.	.	.	.	.
NO REWIND	.	.	.	.	.	O	.	.	.	.	.	.	.	.	.	.
NO REWIND	.	.	.	.	.	O	.	.	.	.	.	.	.	.	.	.
WITH LOCK	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
READ	I	.	.	.	.	I	I	I	I	I	I	I	I	I	I	I
NEXT	.	.	.	.	.	.	.	.	.	I	.	.	I	.	.	.
FIRST	.	.	.	.	.	.	.	.	.	.	.	.	D	.	.	.
LAST	.	.	.	.	.	.	.	.	.	.	.	.	D	.	.	.
PRIOR	.	.	.	.	.	.	.	.	.	.	.	.	D	.	.	.
INTO	I	.	.	.	.	I	I	I	I	I	I	I	I	I	I	I
AT END	I	.	.	.	.	I	I	I	.	I	I	.	I	I	I	I
INVALID KEY	.	.	.	.	.	.	.	.	I	I	.	I	I	B	.	.
FORMAT	.	.	.	.	.	.	D	.	.	.	D	D	D	I	.	R
NEXT MODIFIED	.	.	.	.	.	.	.	.	.	.	.	.	.	B	.	.
SUBFILE	.	.	.	.	.	.	.	.	.	.	.	.	.	B	.	.

DEVICE TYPE	CARD READER	CARD PUNCH	CARD PRINT	CARD PUNCHPRINT	PRINTER	TAPE	DISK SEQ	DISK REL SEQ	DISK REL RANDOM	DISK REL DYNAMIC	DISK IDX SEQ	DISK IDX RANDOM	DISK IDX DYNAMIC	WORK STATION	DISKETTE	FORMATFILE
INDICATORS	.	.	.	.	.	.	.	.	.	.	.	.	.	I	.	.
TERMINAL	.	.	.	.	.	.	.	.	.	.	.	.	.	O	.	.
NO DATA	.	.	.	.	.	.	.	.	.	.	.	.	.	O	.	.
WRITE	.	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
FROM	.	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
INVALID KEY	.	.	.	.	.	.	.	O	O	O	O	O	O	B	.	.
ADVANCING	.	O	O	O	O	.	.	.	.	.	.	.	.	.	.	.
AT END-OF-PAGE	.	.	.	.	O	.	.	.	.	.	.	.	.	.	.	.
FORMAT	.	.	.	.	.	.	D	.	.	.	D	D	D	R	.	R
STARTING	.	.	.	.	.	.	.	.	.	.	.	.	.	O	.	.
ROLLING	.	.	.	.	.	.	.	.	.	.	.	.	.	O	.	.
INDICATORS	.	.	.	.	.	.	.	.	.	.	.	.	.	O	.	.
SUBFILE	.	.	.	.	.	.	.	.	.	.	.	.	.	B	.	.
TERMINAL	.	.	.	.	.	.	.	.	.	.	.	.	.	O	.	.
START	.	.	.	.	.	.	.	O	.	O	O	.	O	.	.	.
KEY	.	.	.	.	.	.	.	O	.	O	O	.	O	.	.	.
INVALID KEY	.	.	.	.	.	.	.	O	.	O	O	.	O	.	.	.
FORMAT	.	.	.	.	.	.	.	.	.	.	D	D	D	.	.	.
REWRITE	.	.	.	.	.	.	O	O	O	O	O	O	O	B	.	.
FROM	.	.	.	.	.	.	O	O	O	O	O	O	O	B	.	.
INVALID KEY	.	.	.	.	.	.	.	.	O	O	.	O	O	B	.	.
FORMAT	.	.	.	.	.	.	.	.	.	.	.	D	D	B	.	.
INDICATORS	.	.	.	.	.	.	.	.	.	.	.	.	.	B	.	.
SUBFILE	.	.	.	.	.	.	.	.	.	.	.	.	.	S	.	.
TERMINAL	.	.	.	.	.	.	.	.	.	.	.	.	.	O	.	.
DELETE	.	.	.	.	.	.	.	O	O	O	O	O	O	.	.	.
INVALID KEY	.	.	.	.	.	.	.	.	O	O	.	O	O	.	.	.
FORMAT	.	.	.	.	.	.	.	.	.	.	.	D	D	.	.	.
USE	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
EXCEPTION/ ERROR	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O

Table 29 (Page 4 of 4). File Structure Support

DEVICE TYPE	CARD READER	CARD PUNCH	CARD PRINT	CARD PUNCHPRINT	PRINTER	TAPE	DISK SEQ	DISK REL SEQ	DISK REL RANDOM	DISK REL DYNAMIC	DISK IDX SEQ	DISK IDX RANDOM	DISK IDX DYNAMIC	WORK STATION	DISKETTE	FORMATFILE
FOR DEBUGGING	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O	O
COMMIT	.	.	.	.	.	.	D	D	D	D	D	D	D	.	.	.
ROLLBACK	.	.	.	.	.	.	D	D	D	D	D	D	D	.	.	.
ACQUIRE	.	.	.	.	.	.	.	.	.	.	.	.	.	O	.	.
DROP	.	.	.	.	.	.	.	.	.	.	.	.	.	O	.	.

Status Key Values and Meanings

Table 30 (Page 1 of 3). Status Key Values and Meanings

Status Key 1 2	Meaning	When Set (OS/400 Exceptions Monitored, Condition Detected)
0	**Successful Completion**	
0	Successful Completion	No error condition occurred during the I-O operation.
1	**At End of File**	
0	End of file	CPF4740, CPF5001, CPF5025.
2	No modified subfile record found (IBM extension)	CPF5037.
2	**Invalid Key**	
1	Sequence error	REWRITE to an indexed file with sequential access and key for REWRITE ≠ key from previous READ, or WRITE to an indexed file with sequential access and key values for succeeding writes are not in ascending sequence.
2	Duplicate key when duplicates are not allowed	CPF4759, CPF5008, CPF5026, CPF5034, CPF5084, CPF5085, or WRITE to an indexed file with sequential access and key values for succeeding writes are not in ascending sequence.
3	No record found	CPF5001, CPF5006, CPF5013, CPF5020, CPF5025.
4	Boundary violation	CPF5006, CPF5018, CPF5021, CPF5043, CPF5272, if organization is not sequential.
3	**Permanent Error**	
0	Permanent Error	CPF4192, CPF5030, CPF5101, CPF5102, CPF5129, CPF5143.
4	Boundary violation	CPF5018, CPF5116, CPF5272, if organization is sequential.

Table 30 (Page 2 of 3). Status Key Values and Meanings

Status Key 1 2	Meaning	When Set (OS/400 Exceptions Monitored, Condition Detected)
9	**Other Errors**	
0	Other errors: • File not found • Member not found • Level check error • Unexpected I-O exceptions	CPF4101 if a USE is applicable for the file. CPF4102 if a USE is applicable for the file. CPF4131. The following exceptions are monitored generically: CPF4101 through CPF4399 CPF4501 through CPF4699 CPF4701 through CPF4899 CPF5001 through CPF5099 CPF5101 through CPF5399 CPF5500 through CPF5699 These exceptions are caught and FILE STATUS is set to 90. If a USE procedure is applicable, it is processed. Otherwise, the program ends and gives the operator the exception and the option to cancel, take a partial dump, or take a full dump.
9	**Other Errors (continued)**	
1	Undefined or unauthorized access type	CPF2207, CPF4104, CPF4236, CPF5057, CPF5109, CPF5134, CPF5279.
2	Logic error: • File locked • File already open • I-O to closed file • READ after end of file • CLOSE on unopened file	CPF4102, CPF4106, CPF4132, CPF4194, CPF4740, CPF5013, CPF5067, CPF5070, CPF5119, CPF5132, CPF5145, CPF5146, CPF5149, CPF5176, CPF5183, CPF5209.
4	No current record pointer	REWRITE/DELETE with sequential access, and last operation was not a successful READ.
5	Invalid or incomplete file information	(1) Duplicate keys specified in COBOL program, but indexed data base file created with unique key; or (2) Duplicate keys not specified in COBOL program, and indexed data base file created allowing duplicate keys.
A	Job has been cancelled in a controlled manner by CL command CNLJOB, TRMSBS, TRMCPF, or PWRDWNSYS	CPF4741. Escape message sent during a READ from invited program device (multiple device files only).
D	Record is locked	CPF5027, CPF5032.

Table 30 (Page 3 of 3). Status Key Values and Meanings

Status Key 1 2	Meaning	When Set (OS/400 Exceptions Monitored, Condition Detected)
9	**Other Errors (continued)**	
H	ACQUIRE operation failed	Resource owned by another program, or unavailable. (9H is the result when an ACQUIRE operation causes any of the OS/400 exceptions monitored for 90 or 9N to occur).
I	WRITE operation failed	CPF4702, CPF4737, CPF5052, CPF5076.
9	**Other Errors (continued)**	
K	Invalid format-name; format not found	CPF5022, CPF5023, CPF5053, CPF5054, CPF5121, CPF5152, CPF5153, CPF5186, CPF5187,
M	Last record written to subfile	CPF5003.
N	Temporary (potentially recoverable) hardware I-O error	CPF4145, CPF4146, CPF4193, CPF4229, CPF4291, CPF4299, CPF4354, CPF4526, CPF4542, CPF4577, CPF4592, CPF4602, CPF4603, CPF4611, CPF4612, CPF4616, CPF4617, CPF4622, CPF4623, CPF4624, CPF4625, CPF4628, CPF4629, CPF4630, CPF4631, CPF4632, CPF4705, CPF5107, CPF5128, CPF5166, CPF5198, CPF5280, CPF5282, CPF5287, CPF5293, CPF5352, CPF5353, CPF5517, CPF5524, CPF5529, CPF5530, CPF5532, CPF5533.
P	OPEN failed because file cannot be placed under commitment control	CPF4285, CPF4293, CPF4326, CPF4327, CPF4328, CPF4329.

Attribute Data Formats

The layouts and values of the attribute data are system dependent. The following formats are for the System/38 environment.

Note: Certain values indicated below can occur only on a System/38 and are not applicable to the AS/400 system.

Display Device Attribute Data

```
 01  TERMINAL ATTRIBUTES.
     02  TERMINAL-TYPE PIC X.
*        D - DISPLAY
     02  TERMINAL-SIZE PIC X.
*        1 - 1920 CHARACTERS
*        2 - 960 CHARACTERS
     02  TERMINAL LOCATION PIC X.
*        L - LOCAL
*        R - REMOTE
     02  TERMINAL-STATUS.
         03  FILLER PIC X.
*            RESERVED
         03  ACQUIRE-STATUS PIC X.
*            Y - ACQUIRED
*            N - NOT ACQUIRED
         03  INVITE-STATUS PIC X.
*            Y - INVITED
*            N - NOT INVITED
         03  DATA-AVAILABLE-STATUS PIC X.
*            Y - DATA IS AVAILABLE
*                (ENTER OR COMMAND KEY PRESSED)
*            N - DATA IS NOT AVAILABLE
     02  FILLER PIC X(9).
*        RESERVED
```

Communications Device Attribute Data

```
 01  COMMUNICATIONS-ATTRIBUTES.
     02  PROGRAM-DEVICE-STATUS PIC X.
*        A - THE PROGRAM DEVICE IS NOT ACQUIRED
*            (VALID FOR LU1, BSC, AND PEER)
*        C - THE PROGRAM DEVICE IS ACQUIRED
*            (VALID FOR LU1, BSC, AND PEER.
*            FOR PEER, THIS IS THE SOURCE END OF THE SESSION.)
*        R - THE PROGRAM DEVICE FOR THE TARGET END OF THE
*            SESSION IS ACQUIRED (VALID ONLY FOR PEER)
     02  INVITE-STATUS PIC X.
*        N - THIS PROGRAM DEVICE IS NOT INVITED
*        I - THIS PROGRAM DEVICE IS INVITED BUT NO DATA
*            HAS BEEN RECEIVED
*        O - THIS PROGRAM DEVICE IS INVITED AND DATA IS READY
     02  SYNCHRONIZATION-LEVEL PIC X.
*        C - SYNVL (*CONFIRM)
*        N - SYNVL (*NONE)
     02  DEVICE-NAME PIC X(10).
*        THE DEVICE NAME ASSOCIATED WITH THE PROGRAM DEVICE
```

OPEN-FEEDBACK and I-O-FEEDBACK Data Areas

OPEN-FEEDBACK

The OPEN-FEEDBACK area is part of the open data path (ODP) that contains information about the OPEN operation. This information is set during OPEN processing and is available as long as the file is open.

This area provides information about the file that the program is using. It contains:

- Information about the file that is currently open, such as:
  - File name
  - File type.
- Information that depends on the type of file that is opened, such as:
  - Printer size
  - Screen size
  - Diskette or tape labels.

I-O-FEEDBACK

The system updates the I-O-FEEDBACK area each time a block of records is transferred between OS/400 and the program. A block of records can contain one or more records.

The I-O-FEEDBACK area is not updated after each read or write for files in which multiple records are blocked and unblocked by COBOL. If the I-O-FEEDBACK area is needed after each read or write in the program, the user can do either of the following:

- Prevent the compiler from generating blocking and unblocking code by not satisfying one of the conditions listed under "Unblocking Input Records and Blocking Output Records" in Chapter 7, "System/38-Compatible COBOL Programming Considerations."
- Specify SEQONLY(*NO) on the Override with Data Base File (OVRDBF) CL command.

Preventing the compiler from generating blocking and unblocking code is more efficient than specifying SEQONLY(*NO).

Even when the compiler generates blocking and unblocking code, certain OS/400 restrictions can cause blocking and unblocking to not be processed. In these cases, a performance improvement will not be realized. However, the I-O-FEEDBACK area will be updated after each read or write.

The I-O-FEEDBACK area contains information about the I-O operation. This area consists of a common area and a device-dependent area. The device-dependent area varies in length and content depending on the file type. This area follows the I-O-FEEDBACK common area and can be obtained by specifying the receiving identifier large enough to include the common area and the appropriate device-dependent area.

The I-O-FEEDBACK area contains information about the last I-O operation, such as:

- Device name
- Device type

- AID character
- Error information for some devices.

See the *System/38 CPF Programmer's Guide.* for a layout and description of the data areas contained in the OPEN-FEEDBACK and I-O-FEEDBACK areas.

Appendix F. EBCDIC and ASCII Collating Sequences

The ascending collating sequences for both the EBCDIC (Extended Binary Coded Decimal Interchange Code) and ASCII (American National Standard Code for Information Interchange) character sets are given in this appendix. Decimal positions within the sequence are given, as well as the binary representation, symbol, meaning for each character, and corresponding decimal position within the other sequence. The symbols with an asterisk * in the right-hand column do not correspond to the same symbol in the other collating sequence.

Note: When using the literal option of the alphabet-name clause, 1 must be added to the number shown in this appendix to specify the corresponding character. (The numbers in this appendix run from 0 to 255; the numbers in the literal option run from 1 to 256.)

EBCDIC Collating Sequence

Collating Sequence	HEX Representation	Binary Representation	Symbol	Meaning	ASCII Number
0	00	00000000	NUL	Null	0
.					
64	40	01000000	Sp	Space	32
.					
74	4A	01001010	¢	Cent sign	91 *
75	4B	01001011	.	Period, decimal point	46
76	4C	01001100	<	Less-than sign	60
77	4D	01001101	(	Left parenthesis	40
78	4E	01001110	+	Plus sign	43
79	4F	01001111	|	Vertical bar, logical OR	33 *
80	50	01010000	&	Ampersand	38
. .					
90	5A	01011010	!	Exclamation point	93 *
91	5B	01011011	$	Dollar sign	36
92	5C	01011100	*	Asterisk	42
93	5D	01011101	)	Right parenthesis	41
94	5E	01011110	;	Semicolon	59
95	5F	01011111	¬	Logical NOT	94
96	60	01100000	–	Minus, hyphen	45
97	61	01100001	/	Slash	47
. .					
106	6A	01101010	¦	Broken vertical bar	124 *

Collating Sequence	HEX Representation	Binary Representation	Symbol	Meaning	ASCII Number
107	6B	01101011	,	Comma	44
108	6C	01101100	%	Percent sign	37
109	6D	01101101	_	Underscore	95
110	6E	01101110	>	Greater-than sign	62
111	6F	01101111	?	Question mark	63
. .					
121	79	01111001	`	Accent grave	96
122	7A	01111010	:	Colon	58
123	7B	01111011	#	Number sign	35
124	7C	01111100	@	At sign	64
125	7D	01111101	'	Apostrophe, prime	39
126	7E	01111110	=	Equal sign	61
127	7F	01111111	"	Quotation mark	34
.					
129	81	10000001	a		97
130	82	10000010	b		98
131	83	10000011	c		99
132	84	10000100	d		100
133	85	10000101	e		101
134	86	10000110	f		102
135	87	10000111	g		103
136	88	10001000	h		104
137	89	10001001	i		105
. .					
145	91	10010001	j		106
146	92	10010010	k		107
147	93	10010011	l		108
148	94	10010100	m		109
149	95	10010101	n		110
150	96	10010110	o		111
151	97	10010111	p		112
152	98	10011000	q		113
153	99	10011001	r		114
.					
161	A1	10100001	~	Tilde	126
162	A2	10100010	s		115

Collating Sequence	HEX Representation	Binary Representation	Symbol	Meaning	ASCII Number
163	A3	10100011	t		116
164	A4	10100100	u		117
165	A5	10100101	v		118
166	A6	10100110	w		119
167	A7	10100111	x		120
168	A8	10101000	y		121
169	A9	10101001	z		122
. .					
192	C0	11000000	{	Left brace	123
193	C1	11000001	A		65
194	C2	11000010	B		66
195	C3	11000011	C		67
196	C4	11000100	D		68
197	C5	11000101	E		69
198	C6	11000110	F		70
199	C7	11000111	G		71
200	C8	11001000	H		72
201	C9	11001001	I		73
. .					
208	D0	11010000	}	Right brace	125
209	D1	11010001	J		74
210	D2	11010010	K		75
211	D3	11010011	L		76
212	D4	11010100	M		77
213	D5	11010101	N		78
214	D6	11010110	O		79
215	D7	11010111	P		80
216	D8	11011000	Q		81
217	D9	11011001	R		82
.					
224	E0	11100000	\	Reverse slant	92
.					
226	E2	11100010	S		83
227	E3	11100011	T		84
228	E4	11100100	U		85
229	E5	11100101	V		86

Collating Sequence	HEX Representation	Binary Representation	Symbol	Meaning	ASCII Number
230	E6	11100110	W		87
231	E7	11100111	X		88
232	E8	11101000	Y		89
233	E9	11101001	Z		90
. .					
240	F0	11110000	0		48
241	F1	11110001	1		49
242	F2	11110010	2		50
243	F3	11110011	3		51
244	F4	11110100	4		52
245	F5	11110101	5		53
246	F6	11110110	6		54
247	F7	11110111	7		55
248	F8	11111000	8		56
249	F9	11111001	9		57
. .					
255	FF				

ASCII Collating Sequence

Collating Sequence	HEX Representation	Binary Representation	Symbol	Meaning	EBCDIC Number
0	00	00000000	NUL	Null	0
. . .					
32	20	00100000	SP	Space	64
33	21	00100001	!	Exclamation point	79 *
34	22	00100010	"	Quotation mark	127
35	23	00100011	#	Number sign	123
36	24	00100100	$	Dollar sign	91
37	25	00100101	%	Percent	108
38	26	00100110	&	Ampersand	80
39	27	00100111	'	Apostrophe, prime	125
40	28	00101000	(	Opening parenthesis	77
41	29	00101001	)	Closing parenthesis	93
42	2A	00101010	*	Asterisk	92
43	2B	00101011	+	Plus	78
44	2C	00101100	,	Comma	107
45	2D	00101101	–	Hyphen, minus	96
46	2E	00101110	.	Period, decimal point	75
47	2F	00101111	/	Slant	97
48	30	00110000	0		240
49	31	00110001	1		241
50	32	00110010	2		242
51	33	00110011	3		243
52	34	00110100	4		244
53	35	00110101	5		245
54	36	00110110	6		246
55	37	00110111	7		247
56	38	00111000	8		248
57	39	00111001	9		249
58	3A	00111010	:	Colon	122
59	3B	00111011	;	Semicolon	94
60	3C	00111100	<	Less-than sign	76
61	3D	00111101	=	Equals	126
62	3E	00111110	>	Greater-than sign	110
63	3F	00111111	?	Question mark	111

Collating Sequence	HEX Representation	Binary Representation	Symbol	Meaning	EBCDIC Number
64	40	01000000	@	At sign	124
65	41	01000001	A		193
66	42	01000010	B		194
67	43	01000011	C		195
68	44	01000100	D		196
69	45	01000101	E		197
70	46	01000110	F		198
71	47	01000111	G		199
72	48	01001000	H		200
73	49	01001001	I		201
74	4A	01001010	J		209
75	4B	01001011	K		210
76	4C	01001100	L		211
77	4D	01001101	M		212
78	4E	01001110	N		213
79	4F	01001111	O		214
80	50	01010000	P		215
81	51	01010001	Q		216
82	52	01010010	R		217
83	53	01010011	S		226
84	54	01010100	T		227
85	55	01010101	U		228
86	56	01010110	V		229
87	57	01010111	W		230
88	58	01011000	X		231
89	59	01011001	Y		232
90	5A	01011010	Z		233
91	5B	01011011	[	Opening bracket	74 *
92	5C	01011100	\	Reverse slant	224
93	5D	01011101	]	Closing bracket	90 *
94	5E	01011110	^ ¬	Circumflex, Logical NOT	95
95	5F	01011111	_	Underscore	109
96	60	01100000	`	Grave accent	121
97	61	01100001	a		129
98	62	01100010	b		130
99	63	01100011	c		131
100	64	01100100	d		132

Collating Sequence	HEX Representation	Binary Representation	Symbol	Meaning	EBCDIC Number
101	65	01100101	e		133
102	66	01100110	f		134
103	67	01100111	g		135
104	68	01101000	h		136
105	69	01101001	i		137
106	6A	01101010	j		145
107	6B	01101011	k		146
108	6C	01101100	l		147
109	6D	01101101	m		148
110	6E	01101110	n		149
111	6F	01101111	o		150
112	70	01110000	p		151
113	71	01110001	q		152
114	72	01110010	r		153
115	73	01110011	s		162
116	74	01110100	t		163
117	75	01110101	u		164
118	76	01110110	v		165
119	77	01110111	w		166
120	78	01111000	x		167
121	79	01111001	y		168
122	7A	01111010	z		169
123	7B	01111011	{	Opening brace	192
124	7C	01111100	|	Vertical line	106 *
125	7D	01111101	}	Closing brace	208
126	7E	01111110	~	Tilde	161

Appendix G. COBOL Reserved Words

The following COBOL reserved word list identifies all reserved words in:

- IBM System/38-Compatible COBOL
- American National Standard COBOL, X3.23-1974
- CODASYL COBOL (from *CODASYL COBOL Journal of Development* dated December 1978).

Each word in the list is preceded by an identifier that is associated with one of the following meanings:

Blank	A System/38-Compatible COBOL reserved word from the 1974 ANS standard.
1	A System/38-Compatible COBOL reserved word that is an IBM extension to the 1974 ANS standard.
2	A COBOL reserved word from the 1974 ANS standard that is not used by System/38-Compatible COBOL. These words should not be used if compatibility is important to an installation. If used, a diagnostic message will be issued.
3	A CODASYL COBOL reserved word that is not included in the 1974 ANS standard and is not supported by System/38-Compatible COBOL as an extension. If used, a diagnostic message will be issued. These words are included for compatibility.

ACCEPT
ACCESS
1 ACQUIRE
ADD
ADVANCING
AFTER
ALL
3 ALPHABET
ALPHABETIC
3 ALPHANUMERIC
3 ALPHANUMERIC-EDITED
ALSO
ALTER
ALTERNATE
AND
3 ANY
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
AUTHOR

BEFORE
3 BEGINNING
3 BINARY
3 BIT
3 BITS
BLANK
BLOCK
3 BOOLEAN
BOTTOM
BY

CALL
CANCEL
2 CD
2 CF
2 CH
CHARACTER
CHARACTERS
2 CLOCK-UNITS
CLOSE
2 COBOL
2 CODE
CODE-SET
COLLATING
2 COLUMN
1 COMMA
1 COMMIT
1 COMMITMENT
3 COMMON
2 COMMUNICATION
COMP
1 COMP-3
1 COMP-4
COMPUTATIONAL
1 COMPUTATIONAL-3
1 COMPUTATIONAL-4
COMPUTE
CONFIGURATION
3 CONNECT
CONTAINS
3 CONTENT
3 CONTINUE
1 CONTROL
1 CONTROL-AREA
2 CONTROLS
3 CONVERSION
3 CONVERTING
COPY
CORR
CORRESPONDING
COUNT
CURRENCY
3 CURRENT

DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
3 DAY-OF-WEEK
3 DB
3 DB-ACCESS-CONTROL-KEY
3 DB-DATA-NAME
3 DB-EXCEPTION
1 DB-FORMAT-NAME
3 DB-RECORD-NAME
3 DB-SET-NAME
3 DB-STATUS
2 DE
DEBUG-CONTENTS
DEBUG-ITEM
3 DEBUG-LENGTH
DEBUG-LINE
DEBUG-NAME
3 DEBUG-NUMERIC-CONTENTS
3 DEBUG-SIZE
3 DEBUG-START
3 DEBUG-SUB
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
3 DEBUG-SUB-ITEM
3 DEBUG-SUB-N
3 DEBUG-SUB-NUM
DEBUGGING
DECIMAL-POINT
DECLARATIVES
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
2 DESTINATION
2 DETAIL
2 DISABLE
3 DISCONNECT
DISPLAY
3 DISPLAY-n
DIVIDE
DIVISION
DOWN
1 DROP
3 DUPLICATE
DUPLICATES
DYNAMIC

2 EGI
ELSE
2 EMI
2 EMPTY
2 ENABLE
END
3 END-ADD
3 END-CALL
3 END-COMPUTE
3 END-DELETE
3 END-DIVIDE
3 END-EVALUATE
3 END-IF
3 END-MULTIPLY
END-OF-PAGE
3 END-PERFORM
3 END-READ
3 END-RECEIVE
3 END-RETURN
3 END-REWRITE
3 END-SEARCH
3 END-START
3 END-STRING
3 END-SUBTRACT
3 END-UNSTRING
3 END-WRITE
3 ENDING
ENTER
ENVIRONMENT
EOP
EQUAL
3 EQUALS
3 ERASE
ERROR
2 ESI
3 EVALUATE
EVERY
3 EXCEEDS
EXCEPTION
3 EXCLUSIVE
EXIT
3 EXOR
EXTEND
3 EXTERNAL
1 EXTERNALLY-DESCRIBED-KEY

3 FALSE
FD
FILE
FILE-CONTROL
FILLER
2 FINAL
3 FIND
3 FINISH
FIRST
FOOTING
FOR
1 FORMAT
3 FREE
FROM

2 GENERATE
3 GET
GIVING
3 GLOBAL
GO
GREATER
2 GROUP

2 HEADING
HIGH-VALUE

HIGH-VALUES

I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
3 INDEX-n
INDEXED
1 INDIC
2 INDICATE
1 INDICATOR
1 INDICATORS
INITIAL
3 INITIALIZE
2 INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

3 KEEP
KEY

LABEL
LAST
3 LD
LEADING
LEFT
2 LENGTH
LESS
2 LIMIT
2 LIMITS
LINAGE
LINAGE-COUNTER
LINE
2 LINE-COUNTER
LINES
LINKAGE
3 LOCALLY
LOCK
LOW-VALUE
LOW-VALUES

3 MEMBER
MEMORY
MERGE
2 MESSAGE
MODE
1 MODIFIED
3 MODIFY
MODULES
MOVE
MULTIPLE
MULTIPLY

NATIVE
NEGATIVE
NEXT
NO
3 NON-NULL
NOT
3 NULL
2 NUMBER
NUMERIC
3 NUMERIC-EDITED

OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
OPEN
OPTIONAL
OR
3 ORDER
ORGANIZATION
3 OTHER
OUTPUT
OVERFLOW
3 OWNER

3 PACKED-DECIMAL
3 PADDING
PAGE
2 PAGE-COUNTER
PERFORM
2 PF
2 PH
PIC
PICTURE
2 PLUS
POINTER
POSITION
POSITIVE
2 PRINTING
1 PRIOR
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
3 PROTECTED
3 PURGE [3]

2 QUEUE
QUOTE
QUOTES

RANDOM
2 RD
READ
3 READY
3 REALM
3 REALMS
2 RECEIVE
3 RECONNECT
RECORD
3 RECORD-NAME
RECORDS
REDEFINES
REEL
3 REFERENCE
3 REFERENCE-MODIFIER
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
3 REPLACE
REPLACING
2 REPORT
2 REPORTING
2 REPORTS
RERUN
RESERVE
2 RESET
3 RETAINING
3 RETRIEVAL
RETURN
REVERSED
REWIND
REWRITE
2 RF
2 RH
RIGHT
1 ROLLBACK
1 ROLLING
ROUNDED
RUN

SAME
SD
SEARCH
SECTION
SECURITY
2 SEGMENT
SEGMENT-LIMIT
SELECT
2 SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
3 SETS
SIGN
SIZE
SORT
SORT-MERGE
2 SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STANDARD-1
3 STANDARD-2
START
1 STARTING
STATUS
STOP
3 STORE
STRING
2 SUB-QUEUE-1
2 SUB-QUEUE-2
2 SUB-QUEUE-3
3 SUB-SCHEMA
1 SUBFILE
SUBTRACT
2 SUM
2 SUPPRESS

2 SYMBOLIC
  SYNC
  SYNCHRONIZED

2 TABLE
  TALLYING
  TAPE
3 TENANT
  TERMINAL
2 TERMINATE
3 TEST
2 TEXT
  THAN
1 THEN
  THROUGH
  THRU
  TIME
  TIMES
  TO
  TOP
  TRAILING
1 TRANSACTION
1 TRUE
2 TYPE

  UNIT
  UNSTRING
  UNTIL
  UP
3 UPDATE
  UPON
  USAGE
3 USAGE-MODE
  USE
  USING

  VALUE
  VALUES
  VARYING

  WHEN
  WITH
3 WITHIN
  WORDS
  WORKING-STORAGE
  WRITE

  ZERO
  ZEROES
  ZEROS

  +
  –
  *
  /
  **
  >
  <
  =

Appendix H. Summary of Clauses and Statements

This appendix contains a summary of all COBOL clauses and statements, giving the page in the manual where the corresponding formats and descriptions may be found.

Table 31. PROCESS Statement

Clause/Statement	Page Reference
PROCESS Statement	46

Table 32. Identification Division

Clause/Statement	Page Reference
IDENTIFICATION DIVISION Statement	267

Table 33. Environment Division

Clause/Statement	Page Reference
ENVIRONMENT DIVISION Statement	269
Configuration Section	270
Input-Output Section	277
FILE-CONTROL Paragraph-Sequential File Entries (READER, PUNCH, PUNCHPRINT, PRINT, PRINTER, TAPEFILE, DISKETTE, FORMATFILE, DISK, DATABASE)	281
FILE-CONTROL Paragraph-Relative (Direct) File Entries (DISK, DATABASE)	282
FILE-CONTROL Paragraph-Indexed File Entries (DISK, DATABASE)	282
FILE-CONTROL Paragraph-Sort or Merge File Entries	283
FILE-CONTROL Paragraph-TRANSACTION File Entries (WORKSTATION)	283
I-O-CONTROL Paragraph	292

Table 34. Data Division

Clause/Statement	Page Reference
FD Entry-Files (FORMATFILE, DATABASE, DISK, READER, PUNCH, PUNCHPRINT, PRINT)	300
FD Entry-Files (DISKETTE)	300
FD Entry-Files (TAPEFILE)	301
FD Entry-Files (PRINTER)	301
FD Entry-TRANSACTION File	302.
SD Entry	303
PICTURE Clause	332

Table 35 (Page 1 of 2). Procedure Division

Clause/Statement	Page Reference
Procedure Division-Format 1-Section, Declaratives	348
Procedure Division-Format 2	348
Class Condition	354
Condition-Name-Condition	355
Relation Condition	356
Sign Condition	356
Switch-Status Conditions	359
Negated Simple Conditions	360
Combined Conditions	360
Abbreviated Combined Relation Conditions	363
Declaratives	364
Exception/Error Declarative	365
ACCEPT Statement	373
ACQUIRE Statement-TRANSACTION File	377
ADD Statement	424
ALTER Statement	452
CLOSE Statement	377
COMMIT Statement	381
COMPUTE Statement	425
DELETE Statement	382
DISPLAY Statement	385
DIVIDE Statement	425
DROP Statement	389
ENTER Statement	468
EXIT Statement	453
GO TO Statement	454
IF Statement	367
INSPECT Statement	429
MOVE Statement	436
MULTIPLY Statement	427
OPEN Statement	389
PERFORM Statement	455
READ Statement	393
REWRITE Statement	402
ROLLBACK Statement	406
SET Statement	440
START Statement	407
STOP Statement	467

Table 35 (Page 2 of 2). Procedure Division

Clause/Statement	Page Reference
STRING Statement	441
SUBTRACT Statement	428
UNSTRING Statement	445
WRITE Statement	412

Appendix I. COBOL Differences

Features of System/38 COBOL Not Supported by System/38-Compatible COBOL

Card devices and card files are not supported on the AS/400 system. Therefore the use of any card-specific language features will cause a severity 30 diagnostic message to be issued for a COBOL compilation in the System/38 environment.

Note: Associated card files cannot be redirected because the compiler uses a single operation (PUTGET) to punch one record and read the next. This operation is valid only to a card device capable of both reading and punching.

On a System/38 the default record sequence for files which allow duplicate keys is FIFO. On the AS/400 system the default is 'no specific sequence'. Users should consider specifying the FIFO keyword in the DDS when creating such files in the System/38 environment, to maintain consistency with files created on a System/38.

Migrating System/38 ANSI 74 COBOL Programs to AS/400 ANSI 85 COBOL/400

This section identifies the changes and conditions that users need to consider when migrating their System/38 ANSI 74 COBOL programs to the AS/400 ANSI 85 COBOL/400 compiler. The following IBM AS/400 ANSI 85 COBOL/400 compiler items are incompatible with the System/38 ANSI 74 COBOL compiler language elements.

- The keyword ALPHABET must precede alphabet-name within the alphabet-name clause of the SPECIAL-NAMES paragraph.
- The relative key data item specified in the RELATIVE KEY phrase must not contain the PICTURE symbol 'P'.
- The ALPHABETIC class test is true for uppercase letters, lowercase letters, and the space character.
- When there is no next executable statement in a called program, an implicit EXIT PROGRAM is processed. (System/38 COBOL defaults to STOP RUN)
- No two files in a MERGE statement may be specified in the SAME AREA or SAME SORT-MERGE AREA clause. The only files in a MERGE statement that can be specified in the SAME RECORD AREA clause are those associated with the GIVING phrase.
- Within the READ the INTO phrase cannot be specified unless:

  All records associated with the file and the data item specified in the INTO phrase are group items or elementary alphanumeric items, or only one record description is subordinate to the file description entry.
- Within the RETURN statement the INTO phrase cannot be specified unless:

  All records associated with the file and data item specified in the INTO phrase are group items or elementary alphanumeric items, or only one record description is subordinate to the sort-merge file description entry.

- File position indicator - the concept of a current record pointer has been changed to a file position indicator.
- Reserved words - new reserved words have been added.
- I/O status - new I/O status values have been added.
- Pseudo-text-1 on the COPY statement must not consist entirely of a separator comma or a separator semicolon.
- A data item appearing in the USING phrase of the Procedure Division header must not have a REDEFINES clause in its data description entry.
- If the FOOTING phrase is not specified, no end-of-page condition independent of the page overflow condition exists.
- The NO REWIND phrase cannot be specified in a CLOSE statement having the REEL/UNIT phrase.
- The CANCEL statement closes all open files.
- When a receiving item is a variable length data item and contains the object of the DEPENDING ON phrase, the maximum length of the item will be used.
- Within the VARYING ... AFTER phrase of the PERFORM statement, identifier-2 is augmented before identifier-5 is set.
- Any subscripts for identifier-4 in the DIVIDE statement REMAINDER phrase are evaluated after the result of the DIVIDE operation is stored in identifier-3 of the GIVING phrase.
- The phrase ADVANCING PAGE and END-OF-PAGE must not both be in a single WRITE statement.
- The picture character-string of an alphabetic item can contain only the symbol "A". There is no editing allowed for the alphabetic data category.
- When a data item described by a PICTURE containing the character "P" is referenced, the digit positions specified by "P" will be considered to contain zeros in the following operations:
  1. Any operation requiring a numeric sending operand
  2. A MOVE statement where the sending operand is numeric and its PICTURE character-string contains the symbol "P"
  3. A MOVE statement where the sending operand is numeric edited and its PICTURE character-string contains the symbol "P" and the receiving operand is numeric or numeric edited
  4. A comparison operation where both operands are numeric.

In addition, other differences in the ANSI 85 COBOL/400 compiler may result in incompatibilities. For example:

- A temporary result field is generated by the compiler although only one identifier is specified, as in the following cases:

  in the ADD statement Format 1, preceding the keyword TO
  in the SUBTRACT statement Format 1, preceding the keyword FROM
  in the MULTIPLY and DIVIDE statements, Format 1.

  This may affect the results of such statements when the identifier in question is also one of the receivers.
- Text word rules are followed for COPY REPLACING.

- On the COPY statement, if text-name has not been qualified, QLBLSRC is assumed as the file name. Also the qualification rules of text-name have changed.
- An index name is a compiler generated storage area that is 4 bytes in length and not 2 bytes in length as in System/38 COBOL.

Note: The length of index data-items has changed from 2 bytes to 4 bytes. Therefore, System/38 or System/38-Compatible COBOL programs that use index data-items as parameters or arguments will require recompilation. Also, files created by System/38 or System/38-Compatible COBOL that include items with usage index will need to be recreated to be used by COBOL/400 programs.

For further information on migrating from System/38 to the AS/400 system, see the *System/38 to AS/400 Migration Aid User's Guide and Reference*.

For further information about COBOL/400 see the *COBOL/400 User's Guide*, and the *COBOL/400 Reference*.

Appendix J. Glossary of Abbreviations

Abbreviation	Stands For	Definition
ANSI	American National Standards Institute	An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.
APPC	Advanced Program-to-Program Communications	Data communications support that allows programs on an AS/400 system to communicate with programs on other systems having compatible communications support. APPC is the AS/400 system method of using the SNA LU6.2 protocol.
APPN	Advanced Peer-to-Peer Networking	Data communications support that routes data in a network between two or more APPC systems that are not directly attached.
ASCII	American National Standard Code for Information Interchange	The standard code used for information interchange between data processing systems, data communications systems, and associated equipment. The code uses a coded character set consisting of 7-bit control characters (8 bits including parity check). The set consists of control characters and graphic characters.
BSC	Binary Synchronous Communications	A data communications line protocol that uses a standard set of transmission and control character sequences to send binary-coded data over a communications line.
DDS	Data Description Specifications	A description of the user's data base of device files that is entered into the system in a fixed-form. The description is then used to create files.
DDM	Distributed Data Management	A function of the operating system that allows an application program or user on one system to use data files stored on remote systems. The system must be connected by a communications network, and the remote systems must also be using DDM.
EBCDIC	Extended Binary-Code Decimal Interchange Code	A coded character set consisting of 256 eight-bit characters.
SEU	Source Entry Utility	The part of the AS/400 system Application Development Tools used by the application programmer to enter and update source and procedure members.
UPSI switch	User Program Status Indicator Switch	A program switch that performs the functions of a hardware switch. Eight switches are provided: `UPSI-0` through `UPSI-7`.

Bibliography

To find out more about Operating System/400 and its control language, refer to:

- *Programming: Control Language Reference, SC41-0030*
- *Programming: Control Language Programmer's Guide*
- *Programming: Reference Summary, SX41-0028*
- *System/36-Compatible COBOL Reference Summary*

For more information on the Screen Design Aid (SDA) utility used to design and code displays, refer to *Application Development Tools: Screen Design Aid User's Guide and Reference, SC09-1340*.

For information on the Source Entry Utility (SEU), refer to *Application Development Tools: Source Entry Utility User's Guide and Reference, SC09-1338*.

For further information on the AS/400 system, refer to the following list of manuals:

- *Communications: Advanced Program-to-Program Communications Programmer's Guide, SC41-8189*, which is intended for the programmer responsible for defining or using OS/400 advanced peer-to-peer networking (APPN). See *Communications: Advanced Peer-to-Peer Networking Guide, SC41-8188* for information on writing application programs that use OS/400 advanced program-to-program communications (APPC).
- *Communications: Distributed Data Management User's Guide* which contains information about remote file processing and distributed data management (DDM) files.
- *Communications: Programmer's Guide*, which provides information an application programmer needs to write application programs that use AS/400 communications and the *Intersystem Communications Function* (OS/400–ICF) file.
- *Callpath/400 Planning and Installation Guide*, GA21-9601, SC21-9601, which provides the following information:
  - Communications information that is common among AS/400 communications support
  - Communication configuration information, such as defining lines, controllers, and devices
  - Information about defining and using display station pass-through
  - Information about the 3270 remote attachment.
- *Information Directory*, which contains a brief description of each manual in the AS/400 library and information on how to order additional publications.
- *COBOL/400 Reference*, SC09-1813, which provides a description of the COBOL/400 language structure, program structure, Procedure Division statements, and compiler directing statements.
- *COBOL/400 User's Guide*, SC09-1812, which provides information to design, write, test, and maintain COBOL/400 programs on the AS/400 system.
- *System/36-Compatible COBOL Reference Summary*, SX09-1287, which summarizes clauses and statements used in System/38-Compatible COBOL.
- *Software Installation*, SC41-3120, which contains step by step procedures to be used when installed licensed programs from IBM.
- *Migrating from System/38 Planning Guide, GC21-9624*, which provides information about migrating products and applications with the System/38 Migration Aid. It includes information for planning the details of migration and performing the functions of the System/38 Migration Aid.
- *Programming: Control Language Programmer's Guide* discusses such AS/400 system programming topics as: objects and libraries; CL programming; message handling; user defined commands and menus; and application testing.
- *Programming: Data Base Guide*, which contains a detailed discussion of the AS/400 system data base structure. This manual also describes how to define files to the system using data description specifications (DDS) keywords.
- *Programming: Data Description Specifications Reference*, which describes the data description specifications (DDS) that are used for describing files.
- *Programming: Data Management Guide*, which contains information about overriding and copying files, describing display, printer, tape, and diskette files to the system, as well as spooling and output queues.
- *Programming: Graphical Data Display Manager Programming Reference*, and *Programming: Graphical Data Display Manager Programming Guide*, which provide guidance on the Graphical Data Display Manager (GDDM[18]) for programmers who need to write graphics applications.

[18] GDDM is a trademark of International Business Machines Corporation

- *Security Concepts and Planning, SC41-8083*, which provides information about general security concepts and planning for security on the system. It also includes information for all users about resource security.
- *System Operator's Quick Reference, SX41-9573*, which describes how to operate the AS/400 system.
- *System/38 Environment Programmer's Guide and Reference* SC41-9755, SC21-9755, which describes migrating from System/38, converting to the AS/400 system, and coexisting in a network.
- *System/38-to-AS/400 Migration Aid User's Guide and Reference, SC09-1165*, which provides information about using the System/38 Migration Aid to move System/38 objects to the AS/400 system using menus and displays, or commands.

Refer to the following System/38 publications for information pertaining to the AS/400 System/38 environment.

- *IBM COBOL Coding Form*, GX28-1464, which is used for coding in the System/38 environment.
- *IBM System/38 Control Program Facility Programmer's Guide*, SC21-7730, which explains how to use CPF commands and data description specifications.
- *IBM System/38 Control Program Facility Reference Manual*, SC21-7806, which describes the data description specifications that are used for describing files.
- *IBM System/38 Control Language Reference Manual*, SC21-7731, which describes commands and parameters that are used for various CPF functions.
- *IBM System/38 Data Communications Programmer's Guide*, SC21-7825., which described commands, parameters, and data description specification keywords that are used for program-to-program and system-to-device communication functions.
- *IBM System/38 Guide to Publications*, GC21-7726., which contains information about System/38 publications, defines terms and lists index entries of frequently used System/38 publications.

Glossary of Terms

abbreviated combined relation condition. The combined condition that results from the explicit omission of a common subject or a common subject and common relational operator in a consecutive sequence of relation conditions.

access method. A method used to read a record from, or to write a record into a file. Access can be sequential (records are referred to one after another in the order in which they appear on the file), it can be random (the individual records can be referred to in a nonsequential manner), or it can be dynamic (records can be accessed sequentially or randomly, depending on the specific request).

access path. The order that records in a data base file are organized for data processing by a program. See also *arrival sequence access path* and *keyed sequence access path*.

actual decimal point. The physical representation of the decimal point position in data using either of the decimal point characters (. or ,). The actual decimal point appears in printed reports and requires a position in storage. Contrast with *assumed decimal point*.

advanced peer-to-peer networking (APPN). Data communications support that routes data in a network between two or more APPC systems that are not directly attached.

advanced program-to-program communications (APPC). Data communications support that allows programs on an AS/400 system to communicate with programs on other systems having compatible communications support. APPC is the AS/400 system method of using the SNA LU6.2 protocol.

ALIAS. An alternative name used by a high level language program to identify (file definition) an object.

alphabet-name. A user-defined word, in the SPECIAL-NAMES paragraph of the Environment Division, which names a character set and/or collating sequence.

alphabetic character. A character that is one of the 26 uppercase characters of the alphabet, or a space.

alphanumeric character. Pertaining to the letters, A-Z; numbers, 0-9; and special symbols, $, #, @, . , or _ .

alphanumeric edited character. An alphanumeric data item with a PICTURE character string that contains at least one B, 0, or /.

American National Standard Code for Information Interchange (ANSCII). The code developed by the American National Standards Institute for information exchange among data processing systems, data communications systems, and associated equipment. The ASCII character set consists of 7-bit control characters and symbolic characters, plus one parity-check bit.

American National Standards Institute (ANSI). An organization sponsored by the Computer and Business Equipment Manufacturers Association for establishing voluntary industry standards.

APPC. See *advanced program-to-program communications (APPC).*

APPN. See *advanced peer-to-peer networking (APPN).*

arithmetic expression. A statement containing any combination of values joined together by one or more arithmetic operators in such a way that the statement can be processed as a single numeric value.

arithmetic operator. A symbol used to represent a mathematical operation, such as: + (addition), – (subtraction), * (multiplication), / (division), and ** (exponentiation).

arrival sequence access path. An access path to a data base file that is arranged according to the order in which records are stored in a physical file. Contrast with *keyed sequence access path.*

ascending key. The values by which data is arranged from the lowest value to the highest value of the key in accordance with the rules for comparing data items. Contrast with *descending key.*

ascending key sequence. The arrangement of data in an order from the lowest value of the key field to the highest value of the key field. Contrast with *descending key sequence.*

assignment-name. A word that associates a file-name with a device.

assumed decimal point. A logical decimal point position that does not occupy a storage position in a data item. It is used by the compiler to align a value properly for calculation or input/output operations. Contrast with *actual decimal point*.

AT END condition. A condition that occurs at the following times: during a READ statement for a sequentially accessed file; during a RETURN statement when no next logical record exists for the associated sort or merge

file; during a SEARCH statement when the search operation ends without satisfying the condition specified in any of the associated WHEN phrases.

auxiliary storage. All addressable storage other than main storage.

binary item. Numeric data that is represented internally as a number in the base 2 numbering system; internally, each bit of the item is a binary digit with the sign as the leftmost bit.

binary synchronous communications (BSC). A data communications line protocol that uses a standard set of transmission and control character sequences to send binary-coded data over a communications line. Contrast with *synchronous data link control*.

block. A group of records that are recorded or processed as a unit.

Boolean data. A category of data items that are limited to a value of 1 or 0.

Boolean literal. A literal composed of a Boolean character enclosed in double quotation marks and preceded by a B; for example, B"1".

bottom margin. A blank area that follows the page body.

boundary violation. An attempt to write beyond the externally defined boundaries of a sequential file.

breakpoint. A place in a program (specified by a command or a condition) where the system stops processing of that program and gives control to the display station user or to a specified program.

BSC file. A device file created by the user to support BSC. Contrast with *communications file.*

called program. A program that is called up and run from within another program (a calling program) or by a command.

calling program. A program that requests the running of another program (a called program).

character. Any letter, number, or other symbol in the data character set that is part of the organization, control, or representation of data.

character constant. The actual character value (a symbol, quantity, or constant) in a source program that is itself data, instead of a reference to a field that contains the data. Contrast with *numeric constant*.

character set. All the valid COBOL characters.

character string. A sequence of characters that form a COBOL word, a literal, a PICTURE character-string, or a comment-entry.

class condition. A condition that specifies that the character content of a data item as all alphabetic or all numeric.

clause. A set of consecutive character-strings that specify a characteristic of an entry. There are three types of clauses: data, environment, and file.

collating sequence. The order in which characters are arranged within the computer for sorting, combining, or comparing.

column. One of two or more vertical sections of printed lines on a page. Each field in the report is a single column.

combined condition. A condition that is the result of connecting two or more conditions with the AND or the OR logical operator.

comment. A remark, criticism, or suggestion about something. A comment is ignored by a compiler.

commitment boundary. Any time there are no outstanding changes for a data base file in a commitment controlled environment.

commitment control. A means of grouping file operations that allows the processing of a group of data base changes as a single unit through the COMMIT command or the removal of a group of data base changes as a single unit through the ROLLBACK command.

common key. The key fields that are common to all record formats in the file starting with the first key field (the most significant) and ending with the last key field (the least significant).

communications file. A device file created by the user to support LU1 SDLC communications. Contrast with *BSC file*.

communications device. A BSC, LU1, or APPC device used through a BSC, communications, or mixed file. In COBOL, these files are defined as ORGANIZATION IS TRANSACTION.

compilation. Translation of a source program (such as RPG or COBOL specifications) into a program in machine language.

compile time. The time during which a source program is translated by a compiler into a machine-language program.

compiler. A program that translates a programming language into a machine language for use by the computer.

compiler-directing statement. A statement that controls what the compiler does, rather than what the compiled program does.

complex condition. A condition in which one or more logical operators (AND, OR, or NOT) act on one or more conditions. Complex conditions include negated simple conditions, combined conditions, and negated combined conditions.

compound condition. A statement that tests two or more relational expressions. The result can be true or false.

computer-name. A system-name that identifies the computer upon which the program is to be compiled or run.

condition. An expression in a program for which a truth value can be determined at run time. Conditions include the simple conditions (relational condition, class condition, condition-name condition, switch-status condition, sign condition) and the complex conditions (negated simple conditions, combined conditions, negated combined conditions).

condition-name. A name assigned to a specific value, set of values, or range of values within the complete set of values that a conditional variable can have.

condition-name condition. A statement that the value of a conditional variable is one of a set (or range) of values assigned to a condition-name associated with the conditional variable.

conditional expression. A simple condition or a complex condition specified in an IF, a PERFORM, or a SEARCH statement. See also *simple condition* and *complex condition*.

conditional statement. A statement that controls program flow based on the result of the evaluation of a condition.

conditional variable. A data item, one or more of which has a condition-name assigned to it.

Configuration Section. A section of the Environment Division of a program which describes the overall specifications of the source and object computers.

connective. A word or a punctuation character that associates a data-name, paragraph-name, condition-name, or text-name with its qualifier; links two or more operands in a series; or forms a conditional expression.

CONSOLE. A function-name associated with the operator's keyboard/display.

constant. A fixed value. See also *literal*.

contiguous items. Consecutive elementary or group items in the Data Division that are contained in a single data hierarchy.

Control Language (CL). The set of all commands with which a user requests system functions.

currency sign. The character $.

currency symbol. The character defined by the CURRENCY SIGN clause in the SPECIAL-NAMES paragraph. If no CURRENCY SIGN clause is present, the currency sign is used. See *currency sign*.

current record. The record that is available in the record area associated with the file.

current record pointer. A method of identifying a record that is used in the sequential processing of the next record.

data base. The collection of all data base files stored in the system.

data clause. A clause in a data description entry in the Data Division that describes a particular characteristic of a data item.

data communications file. A generic term for a communications file or a BSC file. See also *communications file* and *BSC file*.

data description entry. An entry in the Data Division that describes the characteristics of a data item.

data description specifications (DDS). A description of the user's data base or device files that is entered into the system in a fixed-form. The description is then used to create files.

Data Division. One of the four main parts of a COBOL program. The Data Division describes the files to be used in the program and the records contained within the files. It also describes any internal working-storage records that are needed.

data item. A character or a set of consecutive characters (excluding literals in either case) defined as a unit of data by the COBOL program.

data name. A user-defined word that names a data item. When used in the general formats, data name represents a word that cannot be subscripted, indexed, or qualified unless specifically permitted by the rules of that format. See also *identifier*.

debugging line. A COBOL statement run only when the WITH DEBUGGING MODE clause is specified. Debugging lines to help determine the cause of an error.

debugging section. A declaratives section that receives control when an identifier, file-name, or procedure-name is encountered in the Procedure Division.

declarative-sentence. A compiler-directing sentence that specifies when a debugging section or an exception/error procedure is to be run.

declaratives. A set of one or more special-purpose sections at the beginning of the Procedure Division that can be used for input/output error checking or debugging.

delimiter. A character or a sequence of characters that marks the beginning and end of a unit of data but is not part of that unit of data.

descending key. The values by which data is arranged from the highest value to the lowest value of the key field, in accordance with the rules for comparing data items. Contrast with *ascending key*.

descending key sequence. The arrangement of data in order from the highest value of the key field to the lowest value of the key field. Contrast with *ascending key sequence*.

device file. A file that contains a description of how data is to be presented to a program from a device or how data is to be presented to the device from the program. Devices can be display stations, printers, a diskette unit, or a communications line.

digit. Any of the numerals from 0 through 9.

direct file. A data base file in which records are assigned specific record positions by the relative record number. Contrast with *indexed file*.

distributed data management (DDM). A function of the operating system that allows an application program or user on one system to use data files stored on remote systems. The system must be connected by a communications network, and the remote systems must also be using DDM.

division. One of the four major parts in a COBOL program: Identification, Environment, Data, and Procedure.

division header. The reserved words and punctuation that indicate the beginning of one of the four divisions of a COBOL program.

dynamic processing. A method of reading from or writing to a file in a nonsequential order (see *random access*) and reading from a file in a sequential order (see *sequential access*) with the same OPEN statement.

editing character. A single character or a fixed two-character combination that punctuates output.

elementary item. A data item that cannot be further logically subdivided.

entry. An element of information in a table, list, queue, or other organized structure of data or control information.

Environment Division. One of the four main parts of a COBOL program. The Environment Division describes the computers on which the source program is compiled and those on which the object program is run; it also provides a connection between the logical concept of files and their records, and the physical characteristics of the devices on which files are stored.

exception. Something that does not conform to the normal.

exponent. A number, indicating to which power another number (the base) is to be raised. In COBOL, exponentiation is indicated with the symbol ** followed by an exponent.

EXTEND mode. A method of adding records to the end of a sequential file, when the file is opened.

external decimal item. See *zoned decimal item*.

externally described data. Data contained in a file in which the fields in the records are described outside of the program (such as with DDS, IDDU, SQL), and are used by a program when the file is processed. Contrast with *program-described data*.

externally described file. A file in which the record fields are described to the system when the file is created, and used by the program when the file is processed. Contrast with *program described file*.

figurative constant. A reserved word that represents a numeric or character value, or a string of repeated values. The word can be used instead of a literal to represent the value.

FILE-CONTROL. The name and header of an Environment Division paragraph in which the data files for a given source program are named and assigned to specific input/output devices.

file description entry. An entry in the FILE SECTION of the Data Division that contains information about the identification, the physical organization, and the record name of a file.

file-name. A name associated with a file and defined in a file description entry or in a sort-merge file description entry.

file organization. The permanent file arrangement established at the time that a file is created.

FILE SECTION. A section of the Data Division that contains descriptions of all externally stored data (or files) used in a program. Such information is given in one or more file description entries.

file separator. The pages to be produced at the beginning of each output file and used to separate the file from the other files being sent to an output device.

function-name. An IBM-defined name that identifies system logical units, system-supplied information, printer control characters, and program switches.

group item. A named set of consecutive elementary or group items.

hierarchy. A set of entries that includes all subordinate entries to the next equal or higher level number.

Identification Division. One of the four main parts of a COBOL program. In addition to identifying the source program and the object program, this part may also describe the author's name, the location where written, and the date written.

identifier. A data-name that is unique or is made unique by a combination of qualifiers, subscripts, and/or indexes.

imperative statement. A statement that specifies that an action is to be taken unconditionally.

implementer-name. An IBM-defined name that includes assignment names, computer names, function names, and language names.

index. A computer storage position or register, the contents of which identify a particular element in a table.

index data item. A data item in which the contents of an index can be stored without conversion to subscript form.

index name. A user-defined word that names an index.

indexed data name. A data name identifier that is subscripted with one or more index names.

indexed file. A file that records the key and the position of each record in a separate part of the file, called the index.

indexed organization. The file structure that identifies each record by the value of one or more keys within that record.

indicator. An internal switch used by a program to test a field or record or to tell when certain operations are to be performed.

input file. A file from which data is read while the program is running.

input mode. An open mode in which records can be read from the file.

input-output file. A file that is opened in the I-O mode.

Input-Output Section. The section of the Environment Division that names the files and external media needed by an application program. It also provides information required for the sending and handling of data when the program is run.

Input Procedure. A procedure that provides special processing of records when they are released to the sort function.

integer. A positive or negative whole number.

internal decimal item. See *packed decimal item*.

INVALID KEY condition. A run-time condition in which the value of a key for an indexed or direct file does not give a correct reference to the file.

invited device. A display station or communications device that was written to using a DDS format that had the INVITE option specified. For multiple device files, the READ statement will read from any invited program device if no particular program device is specified for input using the TERMINAL phrase and no specific FORMAT is requested.

I-O-CONTROL. The name and the header for an Environment Division paragraph in which program requirements for specific input/output techniques are specified. These techniques include rerun checkpoints, the sharing of same areas by several data files, and the use of a storage-resident cylinder index.

I-O mode. An open mode where records can be read from, written to, or deleted from the file.

job separator. The pages or cards placed at the beginning of the output for each job that has spooled file entries on the output queue. Each separator contains information that identifies the job such as its name, the job user's name, the job number, and the time and date the job was run.

key. A data item that identifies the location of a record, or a set of data items that is used to place data in ascending or descending sequence.

key field. A field used to arrange the records of a particular type within a file member.

key word. A reserved word that is required by the syntax of a COBOL statement or entry.

keyed sequence access path. An access path to a data base file that is arranged according to the contents of key fields contained in the individual records. Contrast with *arrival sequence access path*.

language-name. A system-name that specifies a particular programming language.

level indicator. Two alphabetic characters, FD or SD, that identify the type of file description entry.

level number. A numeric character (1 through 9) or a 2-character set (01 through 49, 66, 77, 88) that begins a data description entry and establishes its level in a data hierarchy. Level-numbers 66, 77, and 88 identify special properties of a data description entry.

library. An object on disk that serves as a directory to other objects. A library is used to group related objects, and allows the user to find objects by name.

library name. A user-defined word that names a library.

LINKAGE SECTION. A section of the Data Division that describes data made available from another program.

literal. A character string whose value is defined by the characters themselves. For example, the numeric literal 7 has the value 7, and the character literal 'CHARACTERS' has the value CHARACTERS. See also *character literal*, *constant*, and *numeric constant*.

local data area. A 1024-byte data area that can be used to pass information between programs in a job. A separate local data area is automatically created for each job.

logical file. A description of how data is to be presented to a program. This type of data base file contains no data, but it defines formats for one or more physical files. Contrast with *physical file*.

logical operator. A reserved word that defines the logical connection between conditions or negates a condition: OR (logical connective–either or both), AND (logical connective–both), and NOT (logical negation).

logical record. The most inclusive data item. The level number for a logical record is 01.

main program. The highest level program involved in a run unit.

main storage. The part of the processing unit where programs are run.

merge file. The temporary file that contains all the records to be merged by a MERGE statement. The merge file is created and can be used only by the merge function.

mixed file. An OS/400 device file that supports: one or more work stations, one or more communications devices, or any combination of work stations and communications devices. A mixed file is processed in COBOL by a file with ORGANIZATION IS TRANSACTION.

mnemonic-name. A user-defined word associated with a function-name in the Environment Division.

mode. See *access method.*

multiple device file. A device file that was created with the maximum number of program devices greater than one. Display files or mixed files can be multiple device files. Contrast with *single device file.*

name. A word that defines a COBOL operand. A name is composed of not more than 30 characters.

native character set. The default character set associated with the computer specified in the OBJECT-COMPUTER paragraph.

native collating sequence. The default collating sequence associated with the computer specified in the OBJECT-COMPUTER paragraph.

negated combined condition. The NOT logical operator immediately followed by a combined condition in parentheses.

negated simple condition. The NOT logical operator immediately followed by a simple condition.

nest. To incorporate a structure or structures into a structure of the same kind; for example, one call instruction (or nested call) within another call instruction (nesting call) or one subroutine (nested subroutine) within another subroutine (nesting subroutine).

next executable statement. The statement to which control is transferred after the current statement has finished running.

next record. The record that logically follows the current record of a file.

noncontiguous item. A data item in the Working-Storage and Linkage Sections of the Data Division that bears no relationship to other data items.

nonnumeric item. A data item that is alphanumeric, alphabetic, or Boolean.

nonnumeric literal. A character string bounded by quotation marks, which literally means itself. See also *literal*.

numeric character. Any one of the digits 0 through 9.

numeric constant. The actual numeric value to be used in processing, instead of the name of a field containing the data. A numeric constant can contain any of the numeric digits 0 through 9, a sign (plus or minus), and a decimal point. Contrast with *character literal*.

numeric edited item. A numeric item whose PICTURE character-string contains valid editing characters.

numeric item. A data item that must be numeric. If signed, the item can also contain a representation of an operational sign.

object program. A set of instructions in machine runnable form. The object program is produced by a compiler from a source program.

OBJECT-COMPUTER. The name of an Environment Division paragraph in which the computer upon which the program will be run is described.

open mode. The condition of a file after the program processes an OPEN statement for that file and before the program processes a CLOSE statement for that file. The particular open mode is specified in the OPEN statement as either INPUT, OUTPUT, I-O, or EXTEND.

operand. The object of a verb or an operator; that is, an operand is the data or equipment governed or directed by a verb or operator.

operational sign. An algebraic sign associated with a numeric data item or a numeric constant that indicates whether the item is positive or negative.

optional word. A reserved word included in a specific format only to improve the readability of a COBOL statement or entry.

output file. A file that is opened in either output mode or extend mode.

output mode. An open mode in which records can be written to a file.

OUTPUT PROCEDURE. A procedure that provides special processing of records when they are returned from the sort or merge function.

overflow. A condition that occurs when a portion of the result of an operation exceeds the capacity of the intended unit of storage.

overlay. To write over (and therefore destroy) an existing file.

packed decimal format. Representation of a decimal value in which each byte within a field represents two numeric digits except the rightmost byte, which contains one digit in bits 0 through 3 and the sign in bits 4 through 7. For all other bytes, bits 0 through 3 represent one digit; bits 4 through 7 represent one digit. For example, the decimal value +123 is represented as 0001 0010 0011 1111. Contrast with *zoned decimal format*.

packed decimal item. A numeric data item that is represented internally in packed decimal format.

paragraph. In the Procedure Division, a paragraph-name followed by a period and a space and by zero, one, or more sentences. In the Identification and Environment Divisions, a header followed by zero, one, or more sentences.

paragraph header. A reserved word, followed by a period and a space that indicates the beginning of a paragraph in the Identification and Environment Divisions.

paragraph name. A user-defined word that identifies and begins a paragraph in the Procedure Division.

parameter. A variable or a constant that is used to pass values between calling and called programs.

phrase. An ordered set of one or more consecutive COBOL character-strings that forms part of a clause or a Procedure Division statement.

physical file. A description of how data is to be presented to or received from a program and how data is actually stored in the data base. A physical file contains one record format and one or more members. Contrast with *logical file*.

physical record. A unit of data that is moved into or out of the computer. Same as *block*.

procedure. One or more successive paragraphs or sections within the Procedure Division, which direct the computer to perform some action or series of related actions.

Procedure Division. One of the four main parts of a COBOL program. The Procedure Division may contain instructions for solving a problem. The Procedure Division may contain imperative-statements, conditional statements, paragraphs, procedures, and sections.

procedure name. A paragraph name or a section name in the Procedure Division.

process. A systematic sequence of operations to produce a specified result.

program-described data. Data contained in a file for which the fields in the records are described in the program that processes the file. Contrast with *externally described data*.

program-described file. In System/38-Compatible COBOL, a file that does *not* have any COPY statement, DDS format, coded as part of the record description entry for the file. The fields in the file's records are described only in the program that processes the file. Contrast with *externally described file*.

program device. A symbolic device that a program uses instead of a real device (identified by the device name). When the program uses a program device, the system redirects the operation to the appropriate real device.

program name. A user-defined word that identifies a COBOL source program.

pseudo-text. A sequence of character-strings and/or separators bounded by, but not including, pseudo-text delimiters. Pseudo-text is used in the COPY REPLACING statement for replacing text strings.

pseudo-text delimiter. Two equal signs (==) used to define the beginning and end of pseudo-text.

punctuation character. A character used to separate COBOL elements or to identify a particular type of COBOL element: a comma, semicolon, period, quotation mark, left or right parenthesis, or space.

qualified data-name. An identifier that is composed of a data-name followed by one or more sets of either of the connectives OF or IN followed by a data-name qualifier.

qualifier. In data processing, all names in a qualified name other than the far right which is called the simple name.

random processing. A method of processing in which specific records can be read from, written to, or deleted from a file order requested by the program that is using them.

read-from-invited-program-devices operation. An input operation that waits for input from any one of the invited program devices for a user-specified time. Contrast with *read-from-one-program-device operation*.

read-from-one-program-device operation. An input operation that will not complete until the specified device has responded with input. Contrast with *read-from-invited-program-devices operation*.

record. A collection of related data or words, treated as a unit; such as one name, address, or telephone number. See also *logical record*.

record area. A storage area in which a record described in a record description entry in the File Section is processed.

record description entry. The total set of data description entries associated with a particular record.

record key. A key field whose contents identify a record within an indexed file.

relational character. One of the characters that expresses a relationship between two operands: = (equal to), > (greater than), < (less than).

relational condition. A condition that relates two arithmetic expressions and/or data items.

relational operator. A reserved word, a relational character, a group of consecutive reserved words, or a group of consecutive reserved words and relational characters used to construct a relational condition.

relative file. See *direct file*.

relative key. An unsigned number that can be used directly by the system to locate a record in a file. Same as *relative record number*.

relative organization. The file organization in which each record is uniquely identified by a positive number value that specifies the position in the file relative to the first record.

relative record number. A number that specifies the location of a record in relation to the beginning of a data base file member, or subfile. For example, the first record in a data base file, member, or subfile has a relative record number of 1.

reserved word. A special word that has a specific meaning to the system as defined in a programming language.

routine. A set of statements in a program that causes the system to perform an operation or a series of related operations.

run time. The time during which the instructions of a computer program are processed by a processing unit.

run unit. A set of one or more programs that run as a set to solve a problem. A set starts with the first COBOL program in the program stack and includes all programs

(COBOL) (non-COBOL) that are below it in the program stack.

section. A set of zero, one, or more paragraphs or entries, called a section body, preceded by a section header. Each section consists of the section header and the related section body.

section header. A combination of words, followed by a period and a space, that indicates the beginning of a *section* in the Environment, Data, or Procedure Division.

section-name. A user-defined word that names a section in the Procedure Division.

sector. An area on a disk track or a diskette track to record information.

sentence. A unit of self-contained text.

separator. A punctuation character used to set apart character strings. See also *file separator* and *job separator*.

sequential access. A method of reading from, writing to, or removing records from a file based on the way the records are arranged in the file.

sequential processing. A method of processing in which records are read, written to, or deleted in the order determined by the value of the key field.

serial search. A search in which the records of a set of records are consecutively examined, beginning with the first record and ending with the last record.

sign condition. A condition that states that the value of a data item is less than, equal to, or greater than zero.

simple condition. One of the conditions chosen from the set: relation condition, class condition, condition-name condition, switch-status condition, and sign condition.

single device file. A device file created with only one program device defined for it. Printer files, card files, diskette files, tape files, communications files, and BSC files are single device files. Display files and mixed files created with a maximum number of one program device are also single device files. Contrast with *multiple device file*.

sort file. A temporary file that contains all the records to be sorted by a SORT statement. The sort file is created and used by the sort function only.

sort-merge file description entry. An entry in the File Section that describes a sort file or a merge file.

SOURCE-COMPUTER. The name of an Environment Division paragraph describing the computer upon which the source program will be compiled.

source program. A set of instructions that are written in a programming language and must be translated to machine language before the program can be run.

special character. A character that is neither numeric nor alphabetic. Special characters in COBOL include: + - * / = $, . ") (; < >

special-character word. A reserved word that is an arithmetic operator or a relational character.

SPECIAL-NAMES. The names of an Environment Division paragraph and the paragraph itself in which names supplied by IBM are related to mnemonic-names specified by the programmer. In addition, this paragraph can be used to exchange the functions of the comma and the period or to specify a substitution character for the currency sign in the PICTURE string.

special registers. Compiler-generated data items used to store information produced by specific COBOL features (for example, the DEBUG-ITEM special register).

spooled file. A file that hold output data waiting to be printed, or input data waiting to be processed by the program.

standard data format. The format in which data is described as it appears when it is printed, rather than how it is stored by the computer.

statement. An instruction in a program. A statement combines COBOL reserved words and user-defined operands.

storage area. A portion of main storage into which data is read or from which it is written.

subfile. A group of records of the same record format that can be displayed at the same time at a work station. The system sends the entire group of records to the work station in a single operation and receives the group in another operation.

subject of entry. A data-name or reserved word that appears immediately after a level indicator or level-number in a Data Division entry. It serves to reference the entry.

subprogram. A called program. A subprogram is combined with the calling program at run time to produce a run unit and is below the calling program in the program stack.

subscript. A positive number or variable, whose value refers to a particular element in a table.

subscripted data-name. A data name that is made unique with a subscript.

switch-status condition. A condition that states that a switch is currently on or off.

synchronous data link control (SDLC). A form of communications line control that uses commands to control the transfer of data over a communications line.

system name. An IBM-supplied name that uniquely identifies the system.

table. A set of logically consecutive data items that are defined in the Data Division with the OCCURS clause.

table element. A data item that can be referred to in a table.

test condition. A statement that, when taken as a whole, may be either true or false, depending on the circumstances existing at the time the expression is evaluated.

text-name. A user-defined word that identifies library text.

text-word. Any character-string or separator, except the space, in copied COBOL source or in pseudo-text.

TRANSACTION file. An input/output file used to communicate with display stations and/or for intersystems communications.

unary operator. A plus sign (+) or a minus sign (–), that precedes a variable or a left parenthesis in an arithmetic expression, which has the effect of multiplying the expression by +1 or -1, respectively.

user-defined word. A word, required by a clause or a statement, that must be supplied by the user in a clause or statement.

user-name. A type of implementor-name that appears in the VALUE OF clause, and that follows the rules for the formation of a user-defined word.

variable. A name used to represent data whose value can be changed while the program is running by referring to the name of the variable.

verb. A reserved word that expresses an action to be taken by a COBOL compiler or an object program.

word. A written, spoken, or transmitted group of characters.

work station. A device used to transmit information to or receive information from a computer; for example, a display station or printer.

Working-Storage Section. A section-name (and the section itself) in the Data Division. The section describes records and noncontiguous data items that are not part of external files but are developed and processed internally. It also defines data items whose values are assigned in the source program.

zoned decimal format. A format for representing numbers in which the digit is contained in bits 4 through 7 and the sign is contained in bits 0 through 3 of the far right byte; bits 0 through 3 of all other bytes contain 1s (hex F). For example, in zoned decimal format, the decimal value of +123 is represented as 1111 0001 1111 0010 1111 0011. Contrast with *packed decimal format*.

zoned decimal item. A numeric data item that is represented internally in zoned decimal format.

Index

Special Characters

*NORANGE option 255

Numerics

00-99 segment-numbers, formation rules 9
01 level-number 310
  illustration 312
01-49 level-numbers, formation rules 9
02-49 level-number concepts 310
  illustration 312
1974 Standard COBOL
  1975 FIPS COBOL and 3
1975 FIPS COBOL 2
1975 FIPS COBOL flagging
  full 3
  high-intermediate 3
  low 3
  low-intermediate 3
5424 MFCU 533
5424 Multi-Function Card Unit (MFCU) 533
66 level-number
  concepts 312
  formation rules 9
  general description 317
  general format 315
77 level-number
  concepts 312
  formation rules 9
88 level-number
  concepts 312
  formation rules 9
  general description 312
  general format 316

A

abbreviated combined relation condition 363
  examples 363
ACCEPT statement 126, 373
  data transfer 373
  DATE 375
  DAY 375
  for TRANSACTION file attributes 126
  formats 374
  mnemonic-name in 273
  system information transfer 375
ACCEPT statement *(continued)*
  TIME 376
access
  dynamic 382, 396, 414
  sequential 289, 382, 393
ACCESS IS DYNAMIC
  relative key required 288
  WRITE statement 414
ACCESS IS RANDOM
  relative key required 289
  WRITE statement 414
ACCESS IS SEQUENTIAL
  relative key optional with 289
  WRITE statement 414
ACCESS MODE clause 123, 288
  default is SEQUENTIAL 288
  formats 282
access modes 280
  compiler-directing statement 349
  file 123
  FORMAT phrase, for TRANSACTION file 119
  logical file 236
access path 227, 287, 401—402
  arrival sequence 227
  example for indexed files 241
  file processing considerations 244
  for indexed files 2, 287
accessing AS/400 features 21
acquire program device
  See ACQUIRE statement
ACQUIRE statement 127
  format 127
actual decimal point
  specification 339
Add Message Description (ADDMSGD) CL command 542
ADD statement 424
  common phrases 422
  composite of operands 420
  formats 424
ADDBKP, CL command 64
ADDBSCDEVE, CL command 128
ADDCMNDEVE, CL command 128
ADDDSPDEVE, CL command 128
adding functions to external description 228

additional notes on field names 225
additional notes on format names 225
ADDMSGD, CL command 542
ADVANCING phrase 230
  of WRITE statement 412
AFTER ADVANCING phrase
  of WRITE statement 412
AFTER phrase of INSPECT statement 434
algebraic comparison
  relation condition 356
  sign test uses 359
algebraic sign 314
algorithms, compiler 537
alias name 220
  example 218
alignment rules
  alphabetic items 313
  alphanumeric edited items 313
  alphanumeric items 313
  decimal point in arithmetic statements 313
  in an elementary MOVE statement 439
  JUSTIFIED clause modifies 328
  numeric edited items 313
  numeric items 313
ALL 11
ALL literal figurative constant 12
ALL phrase of INSPECT statement 430
ALL PROCEDURES phrase (DEBUGGING) 521
Allocate Object command 209
alphabet-name
  CODE-SET clause specification 309
  formation rules 9
alphabet-name clause 275
  COLLATING SEQUENCE phrase and 275
  format 271
  literal phrase 275
  NATIVE phrase 275
  PROGRAM COLLATING SEQUENCE clause and 275
  STANDARD-1 phrase 275
alphabetic characters 6
  COBOL character set 6
  in CURRENCY SIGN clause 277
ALPHABETIC class test rules 355
alphabetic item
  alignment rules 313
  PICTURE clause considerations 336
alphanumeric character 38, 66
alphanumeric edited item 528
  alignment rules 313
alphanumeric edited item *(continued)*
  PICTURE clause considerations 337
alphanumeric item
  JUSTIFIED clause and 327
  PICTURE clause considerations 337
  RECORD KEY data item 290
  status key 291
ALSO phrase of alphabet-name clause 276
ALTER statement 452
  format 452
  segmentation considerations 453, 506
altered GO TO statement 453, 454
American National Standard COBOL
American National Standard Code for Information Interchange (ASCII) 575
  alphabet-name clause and 275
  COLLATING SEQUENCE phrase and 497
  collating sequences 555
  PROGRAM COLLATING SEQUENCE clause 275
American National Standards Institute (ANSI) 575
AND logical connective 10
  in combined condition 359
AND NOT logical connective 10
ANSI
  See American National Standards Institute (ANSI)
apostrophe
  punctuation character 7
  specified in PROCESS statement 43
  used as quotes 6
  within nonnumeric literal 7
APPC devices 135
application-oriented menu 59
  example of 59
Arabic numeral
  in COBOL character sets 6
Area A, columns 8 through 11 25
Area B, columns 12 through 72 25
arithmetic expression 351, 352
  COMPUTE statement operand 425
  in relation condition 356
  in sign test 359
  in WHEN phrase of SEARCH ALL 483
  operators used 353
arithmetic operations
  combining 425
  order rules 353

arithmetic operators 14, 34, 353
arithmetic statement operands
  overlapping 421
  size of 420
arithmetic statements
  ADD statement 424
  common phrases 422
  COMPUTE statement 425
  CORRESPONDING phrase 422
  DIVIDE statement 425
  GIVING phrase 423
  multiple results 421
  MULTIPLY statement 427
  operands 420
  ROUNDED phrase 423
  SUBTRACT statement 428
arithmetic symbol pair list 354
arrival sequence access path 227
AS/400 features, accessing 21
AS/400 system, differences from System/38 571
ASCENDING/DESCENDING KEY phrase
  of OCCURS clause 478
    formats 476
  SORT/MERGE 496
    length of KEY data item 496
ASCII
  See American National Standard Code for Information Interchange (ASCII)
ASSIGN clause 122, 284
  formats 282
  indicators 93
ASSIGN clause with separate indicator area attribute 93
assigning index values 488
assignment-name
  as function-name 9
  ASSIGN clause 284
    association 286
    attribute 284
    device 284
    formats 281
    hopper 286
    name 286
  RERUN clause 292
associated card files 533
association entry, unique 534
assumed decimal point 313
  alignment in numeric item 313
asterisk (*)
  begins comment line 29
asterisk (*) *(continued)*
  comment line 29
  precedes comment line 13
  source code with 23
AT END condition
  and SEARCH ALL statement 483
  EXCEPTION/ERROR Declarative and 365
  READ statement considerations 134, 138, 396
AT END phrase 135, 138
  of SEARCH statement 483
  status key 370
attention functions 90
attribute data for program device
  formats 551
  obtaining 127
attribute data formats 127
ATTRIBUTE-DATA mnemonic-name
  and ACCEPT statement 127
  formats 551
attributes of the item 53
AUTHOR paragraph, Syntax Checker
  restriction 23
auxiliary storage file 295, 543
availability of records 372

B

batch compiles 47
batch jobs 37
BEFORE ADVANCING phrase
  WRITE statement 412
BEFORE/AFTER phrase of INSPECT statement 434
binary data 122
binary item
  USAGE clause considerations 324
binary operators 353
Binary Synchronous Communications (BSC) 575
bit configuration of hexadecimal digits 325
blank line 30
BLANK WHEN ZERO clause 328
  format 328
  VALUE clause considerations 330
BLOCK CONTAINS clause 303
  format 303
  I-O-FEEDBACK special register and 304
blocking code, generation of 552
blocking output record
  See unblocking input records/blocking output records

blocking output records 210
blocking, automatic 370
Boolean data facilities 126
  See also indicators
  comparison rules 358
  description 92
  format 94, 316
  sending/receiving items 439
Boolean literal
  characters permitted in 8
Boolean literal delimiters (B" and ")
  placement rules for 29
bottom page margin in LINAGE clause 306, 307
boundary
  alignment 327
  commitment 247
brackets, square
  optional 14
  use of 38
breakpoints 63
  considerations 69
  data-name considerations 66
  example 63
browsing through a compiler listing 48
BSC (Binary Synchronous Communications) 575
BSC files
  data organization for 530
  support 90
business problems, processing of 1

C

CALL GDDM
  See graphics support
CALL QCL, CL command 21
CALL statement 510
  control language 59
  dynamic 510
  examples 514
  formats 510
  inter-program communication concepts 510
  ON OVERFLOW phrase 510
  segmentation considerations 506
  static 510
  USING phrase 511
called program
  segmentation considerations 506
calling for HELP 88
calling program
  segmentation considerations 506
Calls between programs 1
CANCEL statement
  example 513
  format 512
  inter-program communication concepts 262
card file processing 533
card files, associated 533
categories
  of data, concepts 312
  of statements 349
CBL9001, escape message 38
Change Debug (CHGDBG) CL command 61
Change Job (CHGJOB) CL command 75
Change Job Description (CHGJOBD) CL command 75
change/date field 51
character codes and CODE-SET clause 309
character set 314
  Character set for COBOL 6
    ascending EBCDIC sequence 6
  IBM extension 6
character string
  alphabetic 6
  and item size 314
  in INSPECT statement 430
  numeric 6
  picture 12
  representation in PICTURE clause 334
  special 6
  used as literal 7
  uses of 7
character-string considerations, IBM extensions 527
characters
  in a user-defined word 9
  meaning of in COBOL 6
  permitted in a numeric literal 8
  permitted in Boolean literal 8
  used in PICTURE clause 332
  valid as separators 13
characters allowed
  user-defined word 9
characters and character strings 6
CHARACTERS phrase
  of BLOCK CONTAINS clause 304
  of INSPECT statement 434
checking syntax 1
CHGCMDDFT, CL command 39
CHGJOB, CL command 75

CHGJOBD, CL command 75
CHGPGMVAR, CL command 69
CL commands
  ADDBKP 64
  ADDBSCDEVE 128
  ADDCMNDEVE 128
  ADDDSPDEVE 128
  ADDMSGD 542
  CALL QCL 21
  CHGCMDDFT 39
  CHGDBG 61
  CHGJOB 75
  CHGJOBD 75
  CHGPGMVAR 69
  CRTCBLPGM 37
  CRTDKTF 309, 528
  CRTDSPF 128
  CRTJOBD 75
  CRTMXDF 128
  CRTTAPF 309
  ENTDBG 61
  JOB 75
  MONMSG 38
  OVRDBF 286
  OVRDKTF 309
  OVRDSPF 128
  OVRMSGF 542
  OVRTAPF 309, 528
  RSMBKP 63
  SBMJOB 75
  STRSEU 21
  user-created 59
class condition
  EBCDIC signs in 355
  format 354
class test rules 355
classes of data, concepts 312
clause
  ACCESS MODE 123
  ASSIGN 122
  CONTROL-AREA 124
  entry 5
  FILE STATUS 124
  independent 125
  INDICATOR 95
  LINAGE 230
  OCCURS 94
  optional 15
  ORGANIZATION 123
  PICTURE 94
clause *(continued)*
  RELATIVE KEY 124
  required 15
  rules for use 5
  USAGE 94
  VALUE 95
clauses and statements, summary of 567
CLOSE statement
  access considerations 378
  device considerations 378
  FOR REMOVAL phrase 377
  for TRANSACTION file 128
  formats 128, 377
  LOCK phrase 377
  organization considerations 378
  REEL/UNIT phrase 378
  volume considerations 378
COBOL 1
  CALL statement 59
  character set
    ascending EBCDIC sequence 6
    IBM extension 6
  coding forms 21
    example 25
  command statement
    options 41
    used to compile a COBOL program 37
  meaning of characters in 6
  program
    divisions of 4
    organizing 4
    required storage 38
  syntax checker
    restrictions on source 22
    used by SEU 22
  word 8
    maximum length 9
    reserved 9, 563
COBOL divisions, functions of 5
CODE-SET clause 309
  format 309
  omission of 309
  specified for diskette files 309
  specified for tape files 309
coding example
  COPY DDS results 218, 224
  Data Division 297
  DDS for a record format 217
  DDS for field reference file 215
  DDS for keyed access path 241

coding example *(continued)*
initialize a table to zero 476
INSPECT statement 433
PERFORM statement 459, 462, 466
Procedure Division 352
SEARCH statement 486
SPECIAL-NAMES paragraph 274
subscripting 473
coding, COBOL, forms 21
coding/entering programs 21
COLLATING SEQUENCE phrase
alphabet-name clause and 275
of SORT/MERGE statements 497
collating sequences
EBCDIC and ASCII 555
user-specified 275
column
sequence error indicator 50
column 7
continuation area 25
D denotes debugging line 524
columns 1 through 6 for sequence numbers 24
combined arithmetic operations 425
combined condition
format 360
combined relation condition, abbreviated 363
examples 363
comma (,)
editing character 6
in Configuration Section 270
in data description entry 317, 318
in File-Control entry 284
in I-O-CONTROL paragraph 292
programming use 6
punctuation character 6
separator, rules for using 13
series connective 10
comma and decimal point, interchanging 277
command summary 49
command, CL
See CL commands
comment
forms of 12
comment line 31, 34
punctuation characters valid in 30
rules 29
successive 29
with asterisk 29
with slash 29
comment-entry 29
as a comment 12
entry 12
in Identification Division 267
use 12
COMMIT statement
format 381
commitment boundary 247
commitment control 255
considerations 247
example program 250
recovery after failure using 257
recovery with 257
COMMITMENT CONTROL clause 293
format 292
common data concepts 309
common keys 228
common phrases, arithmetic statements
CORRESPONDING phrase 422
GIVING phrase 423
ROUNDED phrase 423
SIZE ERROR phrase 423
common processing facilities 130
current record pointer 371
INTO/FROM phrases 371
invalid key condition 371
status key 370
COMMUNICATION module 2
communications
considerations, inter-program 262
inter-program 262
recovery 257
example program 258
communications files
data organization for 530
support 90
comparison rules
Boolean operands 358
INSPECT statement 431
START statement 407
compilation
with a remote AS/400 file 278
WITH DEBUGGING MODE 517
compilation date in source listing 269
compilation statistics 49
compile-time
messages 541
options 46
compiled programs, running 59

compiler
action on intermediate results 537
algorithms 537
features 1
information field 21
listing, browsing through 48
messages 49
temporary result field 421
compiler options 37, 41
listing of 49
overriding 37
retrieving 48
specified on CRTCBLPGM command 37
specified on PROCESS statement 46
compiler output 49
listing descriptions 49
listing examples 50
compiler-generated statement number 50
compiling source programs 37
complex conditions
combined conditions 360
in PERFORM statement 459
negated simple conditions 360
composite of operands 420
ADD statement processing and 424
arithmetic statements 420
SUBTRACT statement processing rules 428
COMPUTATIONAL item
USAGE clause considerations 323
COMPUTE statement 425
format 425
computer-name
as system-name 9
form of 271
concatenating data items 441
concatenation expressions 38
concepts
data description 309
Sort/Merge 491
concepts, inter-program communication
common data 507
control transfers 507
language considerations 508
concepts, segmentation
control 504
fixed segments 503
independent segments 503
logic 504
condition
class 354, 355
condition *(continued)*
combined 360
complex 359
in IF statement 367
INVALID KEY 366, 371
permissible element sequences 361
relation 356, 357, 358, 481
sign 359
simple 354
switch-status 359
condition-name 19, 317
and SET statement 440
condition 355
formation rules 9
qualification format 17
switch-status condition 359
VALUE clause considerations 328
condition-name condition
example 356
format 355
PROGRAM COLLATING SEQUENCE clause 272
condition-name entry 330
concepts 312
general format 316
condition-names, restrictions 19
condition-names, setting to true 529
conditional expressions 351
complex conditions 359
evaluation rules 361
in PERFORM statement 458
simple conditions 354
conditional GO TO statement 455
conditional PERFORM statement 458
conditional statement
categories of 349
IF statement 367
format 367
nested IF statement 368
conditional variable 317
condition-name condition tests 355
condition-name entries 330
FILLER allowed as name 317
Configuration Section
format 270
connectives
types of 10
consecutive statements 20
considerations, system dependent
DATA DIVISION considerations
BLOCK CONTAINS clause 303

considerations, system dependent *(continued)*
DATA DIVISION considerations *(continued)*
COPY DDS statement 219
index literals 474
item size 304
LINAGE clause 307
OCCURS clause 476
RECORD CONTAINS clause 304
SORT/MERGE statement 495
subscript literals 472
ENVIRONMENT DIVISION considerations
ASSIGN clause 284
RECORD KEY clause 289
RESERVE clause 286
SAME AREA or SAME RECORD AREA clause 292
SAME SORT-MERGE AREA clause 493
general considerations
indexed file 288
library-name 17
program-name 268
relative file 288
source program library 30
source statements 22
text-name 17
user-defined words 9
PROCEDURE DIVISION considerations
arithmetic statements 420
CALL statement 510
GO TO DEPENDING ON statement 454
INSPECT statement 429
STOP statement 467
UNSTRING statement 445
constant, figurative 11, 12
contents of DEBUG-ITEM special register 523
continuation area
column 7 25
D denotes debugging line 524
continuation line
rules 29
control flow
PERFORM statement 458
SEARCH ALL statement 483
SEARCH statement 481
Control Language (CL)
CALL statement 59
control of segmentation 504
control return, in PERFORM statement 457
control screen management functions 90
control transfer
changed by ALTER statement 452
inter-program communication concepts 507
PERFORM statement 457
control transfer rules
Declarative procedures 364
explicit, GO TO statement 454
control transfers, explicit and implicit 20
CONTROL-AREA clause 124
control, commitment 255
considerations 247
example program 250
recovery after failure using 257
recovery with 257
conventions
used to represent keywords 10
used to represent optional words 10
used to represent reserved words 10
used to represent user-defined words 9
conversion of data
DISPLAY statement and 385
COPY DDS, use with indicators 130, 223
copy function 22
COPY statement 30
and externally described data 220
and floating point 226
and record description entry 299
data field structures 222
DDS and use of 219
DDS results 218, 223
example 34
EXTERNALLY-DESCRIBED-KEY 220
format 30
phrases 30
REPLACING phrase 33
use with TRANSACTION files 90
COPY statement, format 2 93
COPY, within PROCESS statement 48
copyname 51
CORRESPONDING phrase 422, 440
FILLER items ignored 319
MOVE statement considerations 436
count field in INSPECT statement 430
COUNT IN phrase of UNSTRING statement 447
CR (credit) PICTURE symbol 334
sign control symbol 340
Create Diskette File (CRTDKTF) CL command 309
Create Job Description (CRTJOBD) CL command 75

Create Tape File (CRTTAPF) CL command 309
cross-reference feature 1
cross-reference list 49, 56
CRTCBLPGM command 37
CRTCBLPGM options
'text' 46
*ALL 45
*APOST 43
*ATR 43
*BLANK 46
*DUMP 43
*GEN 42
*H 45
*HI 45
*L 45
*LI 45
*LIBL 41, 44
*LINENUMBER 42
*LIST 43
*MAP 42
*NO 45
*NOATR 43
*NODUMP 43
*NOGEN 42
*NOLIST 43
*NOMAP 42
*NONE 45
*NONUMBER 42
*NOOPTIMIZE 44
*NOOPTIONS 43
*NOPATCH 43
*NORANGE 44
*NORMAL 45
*NOSEQUENCE 42
*NOSOURCE 42
*NOUNREF 44
*NOVBSUM 42
*NOXREF 42, 43
*NUMBER 42
*OPTIMIZE 44
*OPTIONS 43
*OWNER 45
*PATCH 43
*PGM 41
*PGMID 41
*QUOTE 43
*RANGE 44
*SEQUENCE 42
*SOURCE 42
*SRCMBRTXT 46
CRTCBLPGM options *(continued)*
*UNREF 44
*USER 45
*VBSUM 42
*XREF 42, 43
00 45
29 44
DUMP 46
file-name 44
library-name 41, 44
program-name 41
QCBLSRC 41
QGPL 41
QSYSPRT 44
severity-level 44, 45
source-file-member-name 41
source-file-name 41
CRTCBLPGM parameters
FIPS 45
FLAG 45
GENLVL 44
GENOPT 43
ITDUMP 46
OPTION 41
PGM 41
PRTFILE 44
PUBAUT 45
SRCFILE 41
SRCMBR 41
TEXT 46
USRPRF 45
CRTCBLPGM, CL command 37
CRTDKTF, CL command 309, 528
CRTDSPF, CL command 128
CRTJOBD, CL command 75
CRTMXDF, CL command 128
CRTTAPF, CL command 309
currency sign
fixed insertion symbol 340
floating insertion symbol 340
CURRENCY SIGN clause 277
format 271
in PICTURE character-string 333
valid characters 277
current record pointer 371, 527
START statement 407

D

data
binary 122
packed 122
referencing 15
requesting from job stream 60
data alignment
in an elementary MOVE statement 438
nonnumeric items 314
numeric items 313
data area, local 266
and ACCEPT statement 373
and DISPLAY statement 385
LOCAL-DATA mnemonic-name 273
data attribute specification 19
data base changes, synchronizing or canceling 527
data base support
file description entry 300
logical record 295
physical record 295
data base, canceling changes to 530
data categories
PICTURE clause 336
data category of nonnumeric literal 8
data class type 53
data classes, description 312
data communications file 90, 122
data conversion
DISPLAY statement 385
in an elementary MOVE statement 438
numeric items 336
SET statement 489
data description
arithmetic statement operands 422
data description entry
Boolean data 94
general description 314
general formats 314
data description specifications (DDS) 575
and externally described files 211
and FORMATFILE files 231
and multiple device files 112
and program described files 229
example for field reference file 215
example for keyed access path 241
example for record format 217
use of keywords 214
Data Division 5, 30, 125
concepts 295
creation of entries 530
Data Division *(continued)*
data description 309
data-names 16
entries 28
Environment Division 5
example 297
file description entry 125, 298, 300
function of 5
Identification Division 5
inter-program communications concepts 508
order of 5
organization 296
format 315
Procedure Division 5
punctuation in 30
sort/merge considerations 493
table handling considerations
OCCURS clause 476
USAGE IS INDEX clause 480
Data Division map 49, 52
data field structures 222
data format, standard 8, 314
data hierarchies
concepts 309
used in qualification 15
data item
description entry concepts 310
level of 53
data item description entry 94, 314
ADD statement considerations 424
breaking apart 445
concatenating 441
general description 314
general format 296
joining together 441
MOVE statement considerations 436
subject of OCCURS clause 477
SUBTRACT statement considerations 428
data manipulation statements
INSPECT statement 429
MOVE statement 436
STRING statement 441
UNSTRING statement 445
data organization, description 279
data receiving fields (UNSTRING) 446
data record size specification 306
data records
in file on another system 278
DATA RECORDS clause
format 306

data reference, methods of 15
data references 56
data references in Procedure Division 19
data relationships 296
data to be punched, formatting of 534
data to be punched, moving of 534
data transfer
  ACCEPT statement 373
  DISPLAY statement 385
  open file 74
  STRING statement 441
  UNSTRING statement 445
data truncation
  ACCEPT statement 373
  incompatible record lengths 305
  nonnumeric items 314
  numeric items 313
data-count fields in UNSTRING statement 446
data-name 348
  formation rules 9
  indexing 528
  qualification 528
  qualification format 16
  REDEFINES clause specification 319
  restriction on duplications 17
  subscript, definition 472
  subscripting 528
data-name clause 317
  format 318
  order of specification 317
data-names 53
  qualifying 16
DATABASE file considerations 236
DATABASE files, processing methods 236
DATABASE versus DISK files 236
date field 50
date of compilation in source listing 269
DATE-COMPILED paragraph 269
  format 267
  syntax checker restriction 24
DATE-WRITTEN paragraph
  syntax checker restriction 24
DATE, ACCEPT statement 375
DAY, ACCEPT statement 375
DB (debit) PICTURE symbol 334
  and numeric edited items 338
  sign control symbol 339
DB-FORMAT-NAME
  other files 373
  TRANSACTION files 117
DB-FORMAT-NAME special register 130, 373
DDM (Distributed Data Management) 575
DDM files 278
DDS (Data Description Specifications)
  See data description specifications (DDS)
DDS format 37
DDS name 211, 219
DEBUG module, 1974 Standard 2
DEBUG-CONTENTS 523
DEBUG-ITEM special register
  figurative constant length and 12
  format 523
  subfield contents 523
debugging 255
debugging COBOL programs 60
debugging Declaratives, processing of 521
debugging features
  compile-time switch 518
  run-time switch 74, 518
  USE FOR DEBUGGING procedures 521
debugging lines 29, 31, 34
DEBUGGING MODE as compile-time switch 518
decimal item
  packed 323
  zoned 323
decimal length 53
decimal numbers, representation 314
decimal point (.)
  actual 339
  alignment of numeric items 313
  alignment of numeric-edited items 313
  and comma, interchanging 277
  assumed 313
  editing character 6
  in a numeric literal 8
  in elementary MOVE statement 438
  punctuation character 6
DECIMAL-POINT IS COMMA clause 277
  comma and period PICTURE symbols 333
  format 271
Declarative procedures
  common exit point 453
  debugging 521
  MERGE statement 494
  SORT statement 494
Declaratives 347
  EXCEPTION/ERROR 365
  EXCEPTION/ERROR for TRANSACTION file 145
  FOR DEBUGGING 521

Declaratives *(continued)*
general format 364
section requirements when used 347
DECLARATIVES keyword
begins Declaratives 347
begins in Area A 25, 26, 28
decrementing index-name values 490
decrementing operands 459
default attributes are implicit 19
defined field 56
definitions
Boolean literal 8
character string 7
COBOL clause 5
COBOL entry 5
COBOL paragraph 5
COBOL phrase 6
COBOL section 5
COBOL sentence 5
COBOL statement 5
COBOL word 8
figurative constants 11
keywords 10
literal 7
numeric literal 8
separator 13
special character words 11
special registers 10
DELETE statement (input/output) 382
access considerations 382
device considerations 382
format 382
organization considerations 382
with duplicate keys 384
DELIMITED BY ALL phrase (UNSTRING) 446
DELIMITED BY phrase
and STRING statement processing 442
delimiter
for Boolean literal 126
for pseudo-text 13
in INSPECT statement 434
in STRING statement 441
in UNSTRING statement 445
DEPENDING ON phrase of GO TO statement 454
DEPENDING ON phrase of OCCURS clause 477
format 477
DESCEND, DDS keyword 527
descending file considerations 246
DESCENDING KEY phrase of OCCURS clause 478
descriptions
COBOL 1
System/38 COBOL 1
DEV parameter, Override command 527
device
IBM-defined 19
invited
See invited devices
program
See program device
program, accessing information about 530
device dependencies 205
device file
ASSIGN clause and 284
multiple 112
single 112
device independence 206
device-dependent area, length of 552
devices, APPC 135
diagnostic levels 541
diagnostic messages 541
listing 57
severity levels 57
suppressing 1
direct and relative index usage 486
direct indexing, 474
disk device type 543
DISK files
considerations 236
processing methods 236
displacement 53
display device file 122
data description specifications for 90
example program 112
record format 90
subfiles 106
display file 107
display file support 90
DISPLAY phrase of USAGE clause 322
display screen formats
COBOL coding form and 22
DISPLAY statement
format 385
mnemonic-name and 273
display, split-end 48
displaying variables, techniques 67

distributed data management (DDM) 278, 575, 577
DIVIDE statement 425
  format 425
division
  data, overall punctuation rules 30
  environment, overall punctuation rules 30
  identification, overall punctuation rules 30
  procedure, overall punctuation rules 30
division header 27
division operator 353
divisions of a COBOL program 4
  functions of 5
documenting end of procedures 453
dollar sign ($)
  See also currency sign
  editing character 6
  use 6
DROP statement 129
  format 129
dump, formatted 60
duplicate primary keys allowed 529
duplicate record keys, DUPLICATES phrase 289
duplication of data-name, restriction 17
dynamic access
  DELETE statement 382
  READ statement 396
  WRITE statement 414
dynamic access mode
  in WHEN phrase of SEARCH ALL 483
  indexed files 281
  relative files 281
  relative key required 289
dynamic values in a table 475

E

EBCDIC (Extended Binary-Code Decimal Interchange Code) 575
EBCDIC character set
  default for alphabet-name clause 275
  HIGH-VALUE 11
  LOW-VALUE 11
  NATIVE phrase 275
EBCDIC collating sequence
  alphabet-name clause 275
  and sort/merge phrase 496
  character set 6
  COBOL characters 6
  editing characters 6
  list of characters 555
edited item, numeric 313, 336
editing in an elementary MOVE statement 438
editing sign control symbols 339
editing sign, description 314
editing through PICTURE clause 338
elementary item 310
  alignment rules 313
  as subscript 472
  classes and categories 312
  level-number concepts 310
  MOVE statement operand 437
  valid clauses 317
elementary moves 437
ELSE phrase 367
  format 367
  with nested IF statements 368
emulating a System/38 21
END DECLARATIVES keywords
  ends Declaratives 347
END DECLARATIVES, keyword 28
end of procedures, documenting 453
end of processing
  STOP RUN statement 467
ENDCBLDBG, CL command 520
ending file processing 377
ENTCBLDBG, CL command 520
ENTDBG, CL command 61
Enter Debug (ENTDBG) CL command 61
ENTER statement as documentation 468
entering a source program 21
entering code, standard COBOL format 24
entering the source program 21
entering/coding programs 21
entries
  subordinate 16
  successive 28
entry 5
  clause 5
  record description 295, 299, 314, 343
Environment Division 5, 30, 122
  Configuration Section 270
  File-Control entry, sort/merge 492
  File-Control paragraph 281
  function of 5
  I-O-Control entry, sort/merge 493
  I-O-CONTROL paragraph 292
  Input-Output Section 277
  punctuation in 30
  Sort/Merge considerations 493
  SPECIAL-NAMES paragraph 272

equal sign (=)
punctuation character 6
relation character 6
rules for using 13
separator, rules for using 13
use 6
EQUAL TO relational operator
in WHEN phrase of SEARCH ALL 484
error conditions
REWRITE statement considerations 405
error correction, automatic 370
error messages 542
errors
interrelational 22
loop 256
escape message CBL9001 38
evaluation results 361
example programs
See also examples
commitment control 250
error recovery procedure 257
FORMATFILE file creation 231
indexed file creation 190
indexed file updating 192
mixed file creation 113
multiple display file creation 113
relative file creation 197
relative file retrieval 201
relative file updating 199
sequential file creation 185
sequential file updating and extension 188
TRANSACTION file processing 131
TRANSACTION program 146
work station support 145
examples
See also example programs
access path for indexed file 241
breakpoint 63
COBOL formatted dump 74
COBOL program skeleton coding 27
commitment control 247
compiler options listing 49, 50
COPY DDS results 218, 224
COPY statement 34
cross reference listing 56
Data Division 302
Data Division coding 297
Data Division map 52
DDS for a display device file 90
DDS for a record format 217
examples *(continued)*
DDS for a record format with Alias keyword 218
DDS for field reference file 215
DDS for subfiles 109
diagnostic messages listing 57
Environment Division coding 270
error recovery 257
FIPS messages listing 55
fixed insertion editing 340
floating insertion editing 341
FORMATFILE file 231
generic START using a program described file 237
generic START using an externally described file 237
Identification Division coding 268
indicators 96
initialize a table to zero 476
INSPECT statement 433, 435
inter-program communication 514
mixed files 113
multiple display files 113
PERFORM statement 459, 462, 466
Procedure Division 352
Procedure Division coding 349
record description concepts 311
record format specifications 215
REDEFINES clause 319
RENAMES clause 345
ROLLING phrase 141
SEARCH statement 486
simple insertion editing 338
source listing 50
SPECIAL-NAMES paragraph 272
STRING statement 443
subscripting 472
trace 72
UNSTRING statement 450
verb usage by count listing 52
work station application programs 145
zero suppression and replacement editing 342
exception monitoring, MONMSG command 38
EXCEPTION/ERROR Declarative
EXTEND phrase 365
file-name phrase 365
format 365
I-O phrase 365
status key 370

EXCEPTION/ERROR procedure
  CLOSE statement 378
  DELETE statement 384
  MERGE statement 494
  REWRITE statement considerations 405
exception/errors 88
exceptional situations
  SORT statement 494
exceptions
  and status key values 548
  causes 256
  in program, monitoring for 60
exit point rules for performed procedures 456
EXIT PROGRAM statement 513
  CALL statement 508
  format 513
  inter-program communications concepts 514
EXIT statement
  format 453
exiting from System/38 environment 21
explicit and implicit references 19
explicit attribute 19
explicit control transfers 19
  GO TO statement 454
explicit references
  Procedure Division 19
exponentiation operator 353
exponentiation results 353
expressions, arithmetic/conditional 351
expressions, concatenation 38
EXTEND phrase of OPEN statement 391
Extended Binary-Code Decimal Interchange Code (EBCDIC) 575
extensions, summary of IBM 527
external data concepts 295
external decimal item
  See zoned decimal item
external description
  adding functions 228
  overriding functions 228
externally described files 211
  adding COBOL functions 228
  considerations for using 211
  OS/400 37
  overriding COBOL functions 228
  specifications 216
externally described printer file 527
externally described TRANSACTION file 90
EXTERNALLY-DESCRIBED-KEY 233, 290
  and COPY statement, DDS, DD format 219, 225

F

fall through of performed procedures 457
FD (File Description) entry 295, 300
  data division 125
  FILE-CONTROL paragraph 284
  formats 300
  general description 295
  implicit redefinition 319
  LABEL RECORDS clause required 305
  OPEN statement 390
  Sort/Merge 493
FD level indicator 16
features of System/38-Compatible COBOL 1
features, AS/400, accessing 21
Federal Information Processing Standard (FIPS) 2
field
  additional information for compiler 21
  change/date 51
  defined 56
  MSGID 57
  names 56
  references 56
  severity-level 57
field definitions
  on remote system 278
  when compiling programs 278
field names, additional notes 225
field-count field, in UNSTRING statement 446
fields
  floating point 226
  floating point key 226
  input 90
  intermediate result 537
  output 90
  output/input (both) 90
figurative constants 11
  length of 12
file access path considerations 244
file categories
  data base files
    logical 278
    physical 278
  device files 278
file considerations 266
  DATABASE 236
  DISK 236
file creation time 414

File Description (FD) entry
See FD (File Description) entry
file feedback
See OPEN-FEEDBACK mnemonic-name
file label specification 305
file locking
Allocate Object command 209
COBOL 209
lock states 209
shared files 209
file processing
access paths 244
associated card 533
DATABASE 236
DISK 236
example programs 185
feedback information 376
FORMATFILE 229
indexed organization 236
initiating 389
methods 244
PRINTER 229
relative organization 242
sequential organization 243
specific 229
summary 279
file QCBLSRC, record length 21
file recovery
after a failure 257
file recovery 257
with commitment control 257
File Section
general description 298
general formats 299
VALUE clause considerations 329
FILE STATUS clause 124, 291
CLOSE statement 378
DELETE statement 382
formats 282
INVALID KEY condition 370
READ statement 398
REWRITE statement 403
START statement 408
file status information
obtaining 74
related exceptions 548
values 548
file structure support summary 543
File-Control entry
file processing entries 281
File-Control entry *(continued)*
sort/merge considerations 492
TRANSACTION file processing entry 122
FILE-CONTROL paragraph
formats 281
function of 284
file-name
CLOSE statement operand 377
DELETE statement operand 382
formation rules 9
in FD entry 303
OPEN statement specification 390
READ statement considerations 395
SD entry operand 493
SELECT clause operand 284
formats 282
SORT statement operand 495
sort/merge file operand 492
START statement specification 408
file-names 53
file(s) 295
auxiliary storage 543
BSC
data organization for 530
support 90
communications 90, 530
data communications 90, 122
DATABASE versus DISK 236
device 284
externally described
OS/400 37
specifications 216
FORMATFILE 231
indexed 236
input 371, 497
logical 278
mixed, data organization for 530
multiple device 112
organization 244
output 293
physical 278
printer, externally described 527
program described 211, 229
QCBLMSG 542
relative 242
sequential 243
single device 112
source, maximum record length 21
TRANSACTION 89

FILLER keyword 318
order of specification 317
FIPS 525
COBOL 2
flagger
1975 flagging 3
compiler messages 55
levels 3
message number 55
messages 49, 55
standard modules used 2
violations flagged 55
FIRST phrase of INSPECT REPLACING statement 433
FIRST phrase, READ statement 396
fixed insertion symbol 340
symbol
fixed length record
size specification 304
fixed length table 477
fixed page spacing, LINAGE clause 306
fixed portion
segmented program 503
floating insertion editing 340
floating point fields 226
floating point key fields 226
footing area, LINAGE clause 307
format
DD 37
DDS 37
format (record) level structures 221
format names, additional notes 225
FORMAT phrase 135, 137
for TRANSACTION file 130
FORMATFILE files 231
example program 231
formation of user-defined words 9
formats
generation of I-O 224
redefinition 225
special display screen 22
formatted dump, COBOL
contents 74
example 75
forms, COBOL coding 21
FROM identifier phrase
REWRITE statement considerations 404
WRITE statement considerations 416
FROM phrase
ACCEPT statement 374
FROM phrase *(continued)*
RELEASE statement 500
function
combining 533
copy 22
of Data division 5
of Environment division 5
of Identification division 5
of Procedure division 5
function-name
ACCEPT statement 373
as system-name 9
DISPLAY statement operand 385
SPECIAL-NAMES paragraph 272
values 274
WRITE statement 416
function-name-1 clause 272
format 271
function-name-2 clause 273
format 271
switch-status condition and 273
functions
attention 90
control screen management 90
copy 22
spooling 22
functions of COBOL divisions 5
fundamental programming techniques 185

G

GDDM
See also graphics support
externally described files 37
processing and externally described TRANSACTION file 92
testing function 60
generation of I-O formats 224
generic START examples
using a program described file 237
using an externally described file 237
generic START statement 237
GIVING phrase
arithmetic statements 423
SORT/MERGE statements 497
GO TO statement 454
Graphical Data Display Manager
See graphics support
graphics support 517

group level names 222
group moves 439

H

header
  division 27
  paragraph 28
  section 27
hexadecimal digit bit configurations 325
hierarchy
  levels of 17
HIGH-VALUE(S) 11
hyphen (-)
  allowed in user-defined word 9
  in continuation area, meaning 29
  in program-name, conversion of 268
  produced when copying Alias names 220

I

I-O files
  EXCEPTION/ERROR Declarative 365
I-O option of OPEN statement 390
  indexed file considerations 389
  relative file considerations 389
  TRANSACTION file considerations 129
I-O-CONTROL paragraph
  formats 292
  order of clauses optional 292
  sort/merge considerations 493
I-O-FEEDBACK mnemonic-name 552
  and ACCEPT statement 376
  extended file status 124
IBM extensions xviii, 527
IBM-defined device 19
IBM-defined switch 19
Identification Division
  format 267
  function of 5
  punctuation in 30
identifier 348
  ACCEPT statement operand 373
  breaking apart 445
  DISPLAY statement operand 385
  general format 18
  in sign test 359
  INSPECT statement operand 429
  replacing characters in 430
IF statement
  format 367
  nested 368
imperative-statement 349
  categories of 349
implicit attribute 19
implicit control transfers 19
implicit references 19
  Procedure Division 19
IN as qualifier connective 15
incrementing index-name values 490
incrementing operands
  PERFORM VARYING rules 459
indentation, rules 28
indentation, to clarify logic 28
independent clause 125
independent segment 503
  calling and called programs 507
index 473
  qualifier connective 10
INDEX usage 479
index-name 474
  and File Section 480
  assigning values 488
  comparison rules 480
  in PERFORM statement 459
  passing values of, in CALL 509
  rules of formation 9, 479
  SET statement operand 488
  values 488
INDEXED BY phrase
  OCCURS clause 476
    formats 476
  SEARCH statement requirements 482
indexed data item 480
  comparison rules 481
indexed file 236
  File-Control entry 287
    format 281
INDEXED I-O module, 1974 Standard 2
indexed organization 279
indexes
  assigning values 488
  conditional variable 330
indexing 473
  INDEXED BY phrase rules 479
  of data-name 528
  subscripting 18
INDICATOR clause 95

indicator structures 222, 223
indicators 255
and COPY statement 221, 223
and Separate indicator area (SI) attribute 93
associated with function keys 90
Boolean data items 92, 101
definition 92
example programs 96
in record area 93
in separate indicator area 93
INDARA DDS keyword 93
option 92
TRANSACTION file processing 92
used with FORMATFILE files 92
INDICATORS phrase 95, 130
initialization
data items with INSPECT statement 430
DEBUG-ITEM special register 523
example for tables 475
indexed file considerations 389
LINAGE-COUNTER 308
of index 474
of table values 475
input fields 90
input file
current record pointer used 371
for sort/merge 497
INPUT phrase
EXCEPTION/ERROR Declarative 365
of OPEN statement 393
relative file considerations 393
input records 210
input spool 208
Input-Output Section 277
format 278
input/output errors
EXCEPTION/ERROR Declarative and 365
INPUT/OUTPUT PROCEDURE control 500
input/output statements
ACCEPT statement 374
ACQUIRE statement 127
CLOSE statement 378
common input/output phrases 370
DELETE statement 383
DISPLAY statement 385
DROP statement 129
OPEN statement 390
READ statement 393
REWRITE statement 404
START statement 407
input/output statements *(continued)*
WRITE statement 412
insertion editing 338
INSPECT statement 429
ALL literal figurative 12
BEFORE/AFTER phrase 434
coding example 433
comparisons illustration 431
examples 435
formats 429
REPLACING phrase 433
TALLYING phrase 433
INSTALLATION paragraph as documentation 268
format 267
syntax checker restriction 24
integer item
RECORD KEY data item 290
RELATIVE KEY data item 288
status key 291
integer places in an ir, calculation of 538
integers, unsigned numeric 18
inter-program call feature 1
inter-program communication
common data 507
local data area 262
concepts
CALL statement 508
control transfers 507
language considerations 508
Data Division, Linkage Section 508
examples 514
EXIT PROGRAM statement 262, 513
CALL statement 513
file considerations 262
initialization 262
return of control 262
STOP RUN statement 262, 514
USING phrase, CALL statement 511
Inter-Program Communication module 2
1974 Standard 1
interactive communications 89
Interactive Data Base Utilities (IDU) 21
interactive jobs 37
interactive messages 541
intermediate result fields 537
intermediate results
SIZE ERROR phrase and 423
internal data concepts 295

internal decimal item
See packed decimal item
internal name 53
internal representation
operational sign 314
interrelational errors 22
interrelationships between program lines 37
INTO identifier phrase of READ statement 398
INTO phrase of RETURN statement 501
INTO/FROM identifier phrase 371
INVALID KEY condition
actions taken 371
EXCEPTION/ERROR Declarative and 366
statements that recognize 371
INVALID KEY phrase 137, 139
DELETE statement and 382
START statement considerations 407
status key 370
WRITE statement 412
invited devices
definition 133
item, attributes of the 53

J

Job (JOB) CL command 75
job's local data area, transfer to 529
JOB, CL command 75
jobs
batch 37
interactive 37
joining data items together 441
JUSTIFIED clause 327
example of results 328
format 327
VALUE clause considerations 330

K

key
status 370
key fields
common keys and 228
defined by DDS 228
descending keys 246
for indexed files 236
in the record area 383
partial keys 237
RECORD KEY clause 228
record keys and 228
KEY phrase
of OCCURS clause 478
of START statement 408
key sequence
ascending 418, 491
descending 418, 478, 491
keys
common 228
record 228
keyword
DECLARATIVES 28
END DECLARATIVES 28
keywords
representation in manuals 10

L

label processing
OPEN statement 392
READ statement 400
WRITE statement 419
LABEL RECORDS clause
format 305
required entry 305
label specification 305
language concepts, inter-program communication 507
language extension
TRANSACTION file 89
language level
of Communication module 2
of Debug module 2
of Indexed I-O module 2
of Inter-program Communication module 2
of Library module 2
of Nucleus module 2
of Report Writer module 2
of Segmentation module 2
of Sequential I-O module 2
of Sort-Merge module 2
of Table Handling module 2
supported by System/38-Compatible COBOL 1
language structure, description 1
language-name
as system-name 9
in ENTER statement 468
LAST phrase, READ statement 396
left parenthesis
punctuation character 6
separator, rules for using 13

length
maximum, source file record 21
nonnumeric literal 527
length of figurative constant 12
length of names 38
less than (<) character
relation character 6
level check function
externally described files 229
level checking 229
level concepts 310
level indicator 16, 296
as qualifier 17
begins in Area A 26
FD 16
level-66 entry 343
SD 16
level number, unequal 528
level of data item 53
level of diagnostic messages 49
level-01 item
implicit redefinition 319
level-01 records 310
level-02 through -49 item 310
level-66 entry 343
general description 317
general format 315
level-77 entry
general description 316
general format 315
level-77 item
Linkage Section considerations 509
level-88 entry 330
general format 316
level-88 item
VALUE clause considerations 329, 330
level-number 16, 296
01 and 77 begin in Area A 28
02-49, 66, 88 begin in Area A or B 28
concepts 310
illustration 312
format 318
formation rules 9
REDEFINES specifications and 319
unequal allowed 310
levels of hierarchy 17
library
source program 30
test 61
LIBRARY module 2
library-name
and System/38 environment library name 31
LINAGE clause
format 306
LINAGE-COUNTER special register and 308
logical page depth illustrated 308
printer file commands 307
when assigned to PRINTER 230
with WRITE END-OF-PAGE 417
WRITE ADVANCING PAGE statement 416
LINAGE-COUNTER special register 308
qualification rules 18
WRITE statement rules for 416
line advancing
WRITE statement rules 416
line continuation 29
line-number, LINAGE-COUNTER value 308
LINENUMBER 42, 47
lines
blank 30
comment 31
debugging 29, 31
successive comment 29
LINES AT BOTTOM phrase of LINAGE clause 307
LINES AT TOP phrase of LINAGE clause 307
Linkage Section
general description 299
Inter-program Communication feature 508
concepts 508
level-77 and level-01 names unique 312
VALUE clause considerations 329
list, cross-reference 49, 56
listing of compiler options 49
listing, verb usage 49
literal 8
alphabet-name 9
as character string 7
Boolean 7
in relation condition 358
INSPECT statement operand 429
maximum size of nonnumeric 7
Nonnumeric 7
Numeric 7
numeric, characters permitted 8
phrase of alphabet-name clause 275
local data area 266
and ACCEPT statement 373
and DISPLAY statement 385

local data area *(continued)*
LOCAL-DATA mnemonic-name 273
LOCK phrase
CLOSE statement 378
locking by COBOL, file 209
locking by COBOL, record 209
locking, file and record 209
logic of segmentation 504
logical connective 359
meaning and use 359
logical connectives, definition of 10
logical file 278
considerations 240
logical operators 34
logical page positioning
LINAGE-COUNTER and 308
logical page size
LINAGE clause specifies 306
logical record 295
BLOCK CONTAINS CHARACTER, clause and 304
level concepts 310
size specification 304
loop errors 256
exceptions 256
refid-inter.considerations 262
loop, tracing a 255
loops in a program 255
LOW VALUE(S) 11
LOW-VALUE/LOW-VALUES figurative constant 11

M

manual
purpose xvii
margins of pages in LINAGE clause 307
maximum length
data description entry 314
numeric literal 8
of table 477
PICTURE character-string 332
Sort/Merge keys 496
table element 478
VALUE clause initialization 330
maximum number
characters in numeric item 336
delimiters in UNSTRING statement 445
digits in numeric item 336
GO TO statement procedure-names 454
lines on printed page 306
maximum number *(continued)*
Sort/Merge keys 496
maximum record length of source files 21
maximum size of nonnumeric literal 7
maximum value
of an index 474
subscript 472
meaning of characters in COBOL 6
period (.) 6
menu, application-oriented 59
merge
concepts 492
MERGE statement
format 494
phrases 495
segmentation considerations 506
sort/merge OUTPUT PROCEDURE 499
merged records 495
message number, FIPS 55
message reply modes 75
message statistics 58
message, escape, CBL9001 38
messages
compile-time 541
compiler 49
modifying 542
numbers 542
severity codes 541
diagnostic, suppressing 1
FIPS 49, 55
interactive 541
level of diagnostic 49
methods of data reference 15
methods of referencing procedures 15
MFCU (Multi-Function Card Unit) 533
5424 533
minimum size
numeric item 336
minimum value
index 474
subscript 472
minus sign (-)
arithmetic operator 7
editing character 7
floating insertion symbol 340
in numeric literal 8
sign 7
sign control symbol 339
use 7

mixed files
description 122
example program 113
subfiles 106
support 90
mnemonic-name
as qualifier 274
formation rules 9
I-O-FEEDBACK 376
OPEN-FEEDBACK 376
setting on/off 529
monitoring for an exception condition 38
monitoring for exceptions 60
MONMSG, CL commands 38
MOVE statement
CORRESPONDING phrase 436
elementary moves 437
formats 436
group moves 439
summary reference table 440
MOVE statement, implicit
INTO/FROM identifier phrase 371
MSGID field 57
multifunction card unit, 5424 533
multiple device file
description 112
example program 113
MULTIPLE FILE TAPE clause 293
format 292
multiple programs 47
multiple redefinitions allowed 320
multiple results, arithmetic 421
processing rules 421
multiplication operator 353
MULTIPLY statement
formats 427

N

name
field 56
internal 53
length of 38
program-id 50
source 53
user-specified 15
naming rules, object 38
NATIVE phrase
COLLATING SEQUENCE phrase 496
of alphabet-name clause 275
negated combined condition 360
negated simple condition
format 360
negative numeric data
SIGN clause and 326
nested IF statement 368
examples 369
next executable statement 20
NEXT MODIFIED phrase 136
NEXT phrase of READ statement 396
NEXT SENTENCE in IF statement 367
NO DATA phrase 135
format 132
NO REWIND phrase
of CLOSE statement 379
of OPEN statement 390
nonnumeric item
ALL literal figurative constant 11
nonnumeric literal
alphabet-name clause 275
data category of 8
maximum length 527
maximum size of 7
punctuation characters 30
punctuation in 8
NOT logical connective
meaning and use 360
placement in combined condition 360
NOUNREF option 38
NUCLEUS module, 1974 Standard 2
number
FIPS message 55
reference 57
statement 53, 57
numeric 8
integers, unsigned 18
numeric characters
allowed in user-defined word 9
NUMERIC class test rules 355
numeric edited item
alignment rules 313
PICTURE clause 338
numeric first character in program-name 268
numeric item
literal 8
PICTURE clause considerations 336
numeric literal
characters permitted 8
DECIMAL-POINT IS COMMA clause 277
value of 8

O

Object Definition Table (ODT) 38
object naming rules 38
object of OCCURS DEPENDING ON clause 477
object program 267
  and table values 475
  processing suspension (STOP) 467
object time
  See run-time
OBJECT-COMPUTER paragraph 271
  format 271
  PROGRAM COLLATING SEQUENCE clause
    relation conditions 272
    SPECIAL-NAMES paragraph and 272
    syntax checker restriction 24
occurrence number
  index-name 481
  subscript identifiers 472
OCCURS clause 94
  ASCENDING/DESCENDING KEY phrase 478
  DEPENDING ON phrase 477
  fixed-length tables 477
  formats 476
  INDEXED BY phrase 479
  variable-length tables 477
ODT 38
OF qualifier connective 10
omission of optional words 10
OMITTED phrase of LABEL RECORDS 305
ON OVERFLOW phrase
  and STRING statement processing 442
  and UNSTRING processing 448
  CALL statement 510
one operand, varying 459
open file, data transfer 74
OPEN INPUT statement
  indexed file considerations 390
  relative file considerations 390
OPEN OUTPUT statement
  LINAGE clause 307
OPEN statement 129, 389
  access considerations 390
  CLOSE statement 377
  device considerations 390
  for TRANSACTION file 129
  formats 389
  initializes LINAGE-COUNTER 308
  organization considerations 390
  sets current record pointer 371
OPEN-FEEDBACK mnemonic-name 552
  and ACCEPT statement 376
operand length
  arithmetic 14
  logical 14
  relational comparisons 357
operands
  overlapping 421
operation
  read-from-invited-program-devices 133
  read-from-one-program-device 133
operation order for arithmetic expressions 353
operational sign 314
  in an elementary MOVE statement 438
  in class test 355
  in numeric item 336
  S PICTURE symbol specifies 333
  SIGN clause 326
operator
  arithmetic 34
  logical 34
  relational 357
  unary 353
operator response
  ACCEPT statement 373
  STOP statement 467
option
  *NORANGE 255
  NOUNREF 38
option indicators 92
OPTION parameter 37
OPTIONAL phrase of SELECT clause 284
  format 282
optional words
  omission of 10
  representation in manuals 10
options
  compile-time 46
  PROCESS statement 47
OR condition, multiple UNSTRING 446
OR logical connective 10
OR NOT logical connective 10
order of clauses
  I-O-CONTROL paragraph 292
order of paragraphs, Identification Division 268
order of symbols in PICTURE clause 335
ordering records using sort/merge 491
ORGANIZATION clause 123
  default is SEQUENTIAL 286
  formats 282

organization of a COBOL program 4
organization, TRANSACTION 123
OS/400 graphics support
  See graphics support
output device, DISPLAY statement 385
output fields 90
output file
  SAME clause and 293
OUTPUT phrase
  and data base files 390
  EXCEPTION/ERROR Declarative 365
  OPEN statement 390
output procedure for sort/merge 499
output spool 208
output, compiler 49
output/input (both) fields 90
overall punctuation rules 30
overflow condition
  and UNSTRING processing 448
  in a STRING statement 442
overlapping delimiters in UNSTRING 445
Override command, DEV parameter 527
override file command 207
Override Message File (OVRMSGF) command 542
Override with Data Base File (OVRDBF) CL command 286
Override with Diskette File (OVRDKTF) CL command 309
Override with Tape File (OVRTAPF) CL command 309
overriding compiler options 37
overriding functions to external description 228
overriding messages 542
OVRDBF, CL command 286, 527
OVRDKTF, CL command 309, 528
OVRDSPF, CL command 128
OVRMSGF, CL command 542
OVRTAPF, CL command 309, 528

P

packed data 122
packed decimal item
  storage occupied 324
  USAGE clause considerations 323
padding of numeric-edited items 313
padding with spaces
  and DISPLAY statement 385
  in a move 437
  incompatible record lengths 305
padding with spaces *(continued)*
  nonnumeric items 314
page advancing rules, WRITE statement 416
page body, definition 307
page ejecting 29
page end, LINAGE clause specifies 416
page margins in LINAGE clause 307
page overflow
  WRITE END-OF-PAGE considerations 417
PAGE phrase of WRITE ADVANCING statement 416
page positioning
  LINAGE-COUNTER 308
page size, LINAGE clause specifies 306
paragraph 5, 347
  SPECIAL-NAMES 230
paragraph header
  specification 28
paragraph-name, specification 28
paragraph-names 16, 347
  and PERFORM statement 456
  formation rules 9
  GO TO statement 454
  qualification format 16
  restriction on duplication 17
parameter
  OPTION 37
parentheses
  left 6
  right 6
  separators, rules for using 13
partial key, referring to 237
PERFORM statement 455
  coding example 459, 462, 466
  common exit point 457
  conditional PERFORM 458
  equivalent to sort/merge 500
  for table search 486
  formats 455
  initializes index 474
  range of 456
  segmentation considerations 506
  TIMES phrase 458
  VARYING phrase 459
performance considerations 255
performed procedures
  common exit point valid 453, 457
  processing rules 456
period (.)
  editing character 6

period (.) *(continued)*
in Configuration Section 270
in data description entry 317, 318
in File-Control entry 284
in I-O-CONTROL paragraph 292
punctuation character 6
separator, rules for using 13
permanent segment, definition 503
permanent segments
ALTER statement 453
permissible comparisons
relation-condition 356
PGR
See graphics support
phrase 6
ADVANCING 230
AT END 135, 138
FORMAT 130, 135, 137
INDICATORS 95, 130
INVALID KEY 137, 139
modifier of clause or statement 6
NEXT MODIFIED 136
NO DATA 135
REPLACING 32
ROLLING 141
STARTING 141
SUBFILE 131
TERMINAL 131, 135, 137, 139, 140, 144
phrases 6
clause 6
qualifying 15
statements 6
physical file 278
physical page size
logical page size 306
physical record
definition 295
physical record size
BLOCK CONTAINS clause 303
specifications 303
picture character strings 12
PICTURE character-string
DECIMAL-POINT IS COMMA clause 277
item size 314
PICTURE clause 94
character-string representation 334
data categories 336
editing 338
editing sign function 314
fixed insertion editing 339
PICTURE clause *(continued)*
floating insertion editing 340
format 332
simple insertion editing 338
special insertion editing 339
symbol order 335
symbols used 332
VALUE clause considerations 329
zero suppression and replacement 342
plus sign (+)
arithmetic operator 6
editing character 6
in numeric literal 8
sign 6
SIGN clause 326
POINTER phrase
and STRING statement processing 442
and UNSTRING processing 448
positive data, and sign control symbols 340
positive numeric data
SIGN clause 326
unsigned 314
prerequisite publication xvii
Presentation Graphics Routines
See graphics support
primary keys, duplicate allowed 529
printer file, externally described 527
printing, preparation of data 534
PRIOR phrase, READ statement 396
problem determination 86
procedure 347
Declarative 364
EXCEPTION/ERROR 365
for debugging 521
general format 364
procedure branching statement 20
ALTER statement 452
GO TO statement 454
in IF statement 367
PERFORM statement 455
STOP statement 467
Procedure Division 126, 347
arithmetic expressions 352
arithmetic statements 420
conditional expressions 354
conditional statements 367
data manipulation statements 429
data references 19
Declaratives 364
example 349

Procedure Division *(continued)*
function of 5
input/output statements 370
LINAGE-COUNTER 308
organization 348
paragraph-names 16
procedure branching statements 452, 454
punctuation 30
procedure-name 347
ALTER statement operand 453
GO TO statement operand 454
PERFORM statement operand 458
procedure-name references 56
procedures, referencing 15
PROCESS statement 37, 46
compiler options specified in 37, 47
APOST 47
FIPS 47
FLAG 47
GEN 47
GENLVL 47
LINENUMBER 47
LIST 47
MAP 47
NOGEN 47
NOLIST 47
NOMAP 47
NONUMBER 47
NOOPTIONS 47
NOSEQUENCE 47
NOSOURCE 47
NOVBSUM 47
NOXREF 47
NUMBER 47
OPTIONS 47
QUOTE 47
SEQUENCE 47
SOURCE 47
VBSUM 47
XREF 47
format 47
in batch compile environment 49
options 47
rules 47
using COPY within 48
values
processing associated card files 534
processing flow
ALTER statement 452
GO TO statement 454
processing flow *(continued)*
PERFORM statement 456
SEARCH ALL statement 483
SEARCH statement 481, 483
STOP statement 467
processing methods for DATABASE files 236
processing methods for DISK files 236
processing of files, initiating 389
processing results
INSPECT statement examples 432, 435
JUSTIFIED clause examples 328
STRING statement 443
UNSTRING statement examples 450
processing rules
INSPECT statement 431
PERFORM statement 457
ROUNDED phrase 423
SIZE ERROR phrase 423
STRING statement 442
UNSTRING statement 447
USE FOR DEBUGGING procedure 521
processing sequence, performed procedures 457
production libraries 61
program (object)
See object program
PROGRAM COLLATING SEQUENCE clause 272
alphabet-name clause 275
condition-name condition 272
format 271
relation condition 272
SPECIAL-NAMES paragraph 272
program debugging switch 517
program described files 211, 229
considerations for using 211
externally defined by DDS with Create File commands 229
program device
accessing information about 530
ACQUIRE operation 128
ACQUIRE operation failure 128
attributes of 128
invited 133
name
specification (see TERMINAL phrase)
used by last I-O operation (see CONTROL-AREA clause)
obtaining information about 132
release operation 129
status of 128

program exception/errors 88
program exceptions, monitoring for 60
program lines, interrelationships between 37
program loops 255
program running 62
program segments 503
  fixed
    permanent 503
  independent 503
program spacing 28
program stopping points 88
program switch
  ALTER statement 453
program syntax, debugging line 524
program termination 60
program testing 62
program-id name 50
PROGRAM-ID paragraph 268
  format 268
program-name 268
  formation rules 9
program, using same in several jobs 63
programs
  coding and entering 21
  debugging 60
  multiple 47
pseudo-text
  replacement rules 32
pseudo-text delimiter
  (==) separator, rules for using 13
  placement rules for 13
publication, prerequisite xvii
publications, list of related xvii
punctuation character
  defined as separator 13
  enclose nonnumeric literal 8
  with nonnumeric literal 8
punctuation in nonnumeric literal 8
punctuation rules 30
punctuation, as separators 13
purpose of manual xvii

Q

QCBLMSG message file 542
QCBLSRC file, record length 21
qualification 15
  CORRESPONDING phrase rules 422
  explicit 15
  implicit 15
  of UPSI condition-names 274
qualification *(continued)*
  rules 17
qualification of data-name 528
qualifier connective 10
qualifier, definition 15
qualifying data-names 16
qualifying phrases 15
quotation mark (')
  placement rules for 29
  punctuation character 7
  rules for using 13
  specified in PROCESS statement 47
QUOTE(S) 11
quotient, in division 425

R

random access 280
  DELETE statement 382
  indexed files 288
  of subfile records 136
  READ statement 393
  relative files 288
  WRITE statement 414
read from invited program devices
  controlled job termination 133
  invited program device errors 133
  name of program device read
    See CONTROL-AREA clause
  no data available
    See NO DATA phrase
  no program devices invited
    See AT END phrase
  operation 133
  record format name read
    See CONTROL-AREA clause
  time out on wait for data 134
READ statement
  access considerations 396
  device considerations 396
  for TRANSACTION file 131
  formats 393
  INTO identifier phrase 371
  organization considerations 396
  sets current record pointer 371
read-from-one-program-device operation 133
receiving field
  alignment rules and 313
  in group MOVE statement 439
  in STRING statement 442
  in UNSTRING statement 447

receiving field *(continued)*
MOVE statement 437
record
See logical record
RECORD CONTAINS clause 304
format 304
record description entry 295, 314
as RENAMES clause qualifier 343
COPY statement, DDS format 299
sort/merge output file 499
record format
fields 90
record format specifications 214
example 215
use of DDS keywords 214
RECORD KEY
for a format 289
function of in indexed file 281
REWRITE statement 404
START statement 409
RECORD KEY clause 289
format 282
record keys 228
RECORD KEYS, valid 237
record length, source file, maximum 21
record level concepts 310
record locking
by COBOL 209
updating data base records 209
record sequencing using sort/merge 491
record size
ACCEPT statement 373
DISPLAY statement 385
established at file creation time 414
incompatible lengths 305
RECORD CONTAINS clause specifies 304
REWRITE statement considerations 402
sort/merge output file considerations 498
record storage, WRITE statement 414
record-description level-number concepts 311
illustration 312
record-name
formation rules 9
RELEASE statement operand 500
REWRITE statement 403
WRITE statement considerations 413
records
blocking output 210
input 210
synchronizing changes to 530
RECORDS phrase of RERUN clause
format 292
REDEFINES clause 319
format 319
redefining item 320
moving to a redefined item 437
redefinition of formats 225
redefinition, group level name 222
reference number 50, 57
reference to data 16
references
data 56
explicit 19
field 56
implicit 19
procedure-name 56
referencing
data 15
procedures 15
referring to a partial key 237
register 523
relation condition
format 356
nonnumeric operand comparisons 358
numeric operand comparisons 358
PROGRAM COLLATING SEQUENCE clause 272
relational operator meanings 357
table handling rules 481
relation condition, abbreviated combined 363
examples 363
relational operator
in abbreviated combined relation condition 363
meaning 357
relative and direct index usage 486
relative file
definition 242
File-Control entry 281
format 283
organization, description 280
relative indexing, 474
RELATIVE KEY
FILE-CONTROL paragraph considerations 284
START statement 409
SUBFILE phrase considerations 131
WRITE statement considerations 416
RELATIVE KEY clause 124

release program device
See DROP statement
RELEASE statement
format 500
releasing a record read for update 210
REMAINDER phrase of DIVIDE statement
format 426
processing rules 426
RENAMES clause
data-name-2 phrase 343
data-name-2 THRU data-name-3 phrase 343
format 343
general format 315
level-66 item 312
specification examples 345
repetitive processing of PERFORM statement 458
replacement editing 342
replacement of file records 402
replacement rules for library-text 31
REPLACING phrase
of COPY statement 33
of INSPECT statement 433
processing 33
REPLACING, with format 2 COPY 226
reply list, system 75
reply modes, message 75
Report Writer module 2
reporting COBOL problems 88
representation
of keywords in manuals 10
of optional words in manuals 10
of reserved words in manuals 10
of user-defined words in manuals 9
RERUN clause 292
formats 292
RESERVE clause 282
reserved word
in COBOL 9
list 563
printed in capital letters 14
representation in manuals 10
types 10
connectives 10
figurative constants 10
key 10
optional 10
special registers 10
special-character 10
use of 10
restrictions of condition-names 19
restrictions on indexing 18
restrictions on subscripting 18
retrieving compiler options 48
retrieving/saving source entries 30
return of control from called program 262
RETURN statement for sort/merge 501
format 501
sets current record pointer 371
REVERSED phrase of OPEN statement 390
REWRITE statement 138, 402
access considerations 403
device considerations 403
for TRANSACTION file 138
format 402
FROM identifier phrase and 371
organization considerations 403
right parenthesis
punctuation character 7
rules for using 13
right-padding of items 313
ROLLBACK statement 406
format 406
sets current record pointer 371
ROLLING phrase 141
ROUNDED phrase 423
ADD statement 423
COMPUTE statement 423
DIVIDE statement 423
MULTIPLY statement 423
processing rules 423
routine-name
formation rules 9
in ENTER statement 468
SUBTRACT statement 423
RSMBKP, CL commands 63
rules
overall punctuation 30
PROCESS statement 47
punctuation 30
rules for forming user-defined words 9
rules for qualification 17
rules for using separators 13
run status, status key usage 291
run unit
CALL statement transfer control 508
run-time
debugging switch 518
run-time switches, debug 74

running a program 62
compiler output 49
methods 59
running the compiler program 59

S

SAME clause 292
and SAME SORT/SORT-MERGE AREA clause 493
format 292
RELEASE statement and 500
saving machine time 1
SBMJOB, CL command 75
SD entry 493
and MERGE statement file-name 494
and RELEASE statement record-name 500
and RETURN statement file-name 501
and SORT statement file-name 494
FILE-CONTROL paragraph 284
format 493
SD level indicator 16
search for next modified record 137
SEARCH statement 481
coding example 486
formats 481
processing considerations 483
section 53
section header 5, 27, 347
in Declarative procedures 364
specification of 27
section-name 5, 16, 347
ALTER statement and 452
as qualifier 17
formation rules 9
GO TO statement 454
PERFORM statement 457
restriction on duplication 17
SECURITY paragraph
format 267
syntax checker restriction 24
SEGMENT-LIMIT clause 272
segment-number 503
formation rules 9
in Declaratives 364
logic of specification 504
segmentation 255
segmentation feature
concepts
program segments 503
Procedure Division 505
segmentation feature *(continued)*
special considerations 506
segmentation information
ALTER statement 506
calling and called programs 507
GO TO statement 506
PERFORM statement 506
SORT and MERGE statements 506
SEGMENTATION module, 1974 Standard 2
SELECT clause
formats 282
order of specification 284
sort/merge considerations 492
sending field
in group MOVE statement 439
in STRING statement 442
in UNSTRING statement 445
MOVE statement 437
sentence 5, 347
categories 349
sentences, successive 28
SEPARATE CHARACTER phrase of SIGN clause
description and format 326
separate indicator area attribute, ASSIGN clause with 93
separators
characters used as 13
IBM extension 13
rules for using 13
sequence
ascending 418, 491
collating 275, 555
descending 418, 491
sequence error indicator column 50
sequence number 24
sequence of processing, performed procedures 457
sequencing records using sort/merge 491
sequential access
DELETE statement 382
of subfile records 136
READ statement and 393
relative key option for 289
WRITE statement 414
sequential access mode 280
indexed files 288
relative files 288
sequential files 281
sequential file 281
definition 243

sequential file *(continued)*
EXCEPTION/ERROR Declarative phrases 365
FILE-CONTROL Paragraph 281
format 281
OPEN statement considerations 389
organization 279
READ statement and 396
REWRITE statement 403
SEQUENTIAL I-O module, 1974 Standard 2
series connective, definition of 10
SET statement
and conditional variables 440
and external switches 440
and multidimensional table search 486
formats 440, 488
initializes index 474
TO phrase 488
UP/DOWN BY phrase 490
valid field combinations 489
SEU
See Source Entry Utility (SEU)
severity level, changing 542
severity levels, message 541
severity of messages 542
severity-level field 57
sharing storage
file records 292
SI attribute 93, 286
sign
currency 340
in a numeric literal 8
operational 314, 326, 333, 336, 355, 438
SIGN clause 326
format 326
minus (-) 7
operational sign representation 314
plus (+) 6
S PICTURE symbol 333
sign condition
format 359
sign control, fixed insertion editing 340
SIGN IS SEPARATE CHARACTER clause
description and format 326
signed decimal items, conversion to 529
signed numeric item
SIGN clause specification 326
simple condition
negation of 360
simple insertion editing 338
single device files 112
SIZE ERROR phrase 423
COMPUTE statement 423
DIVIDE statement 423
MULTIPLY statement 423
SUBTRACT statement 423
size of DEBUG-CONTENTS 523
size of operands in nonnumeric comparisons 358
SIZE, STRING statement delimiter 441
slash (/)
comment line 29
source code with 23
SORT statement
format 495
phrases 496
segmentation considerations 506
sort/merge OUTPUT procedure 500
SORT-MERGE module, 1974 Standard 2
Sort/Merge
concepts 491
considerations 501
Data Division–SD entry 493
Environment Division 492
File Description (SD) entry 493
FILE-CONTROL paragraph required for 284
format 493
File-Control entry 491
I-O-Control entry 493
or AS/400 logical file support 491
Procedure Division
MERGE statement 494
RELEASE statement 500
RETURN statement 501
SORT statement 495
SORT/MERGE statement phrases 496
source code
with asterisk 23
with slash 23
Source Entry Utility (SEU) 575
browsing a compiler listing 48
entering a source program 22
source files 21
source language debugging
compile-time switch 518
DEBUG-ITEM special register 523
debugging lines 524
run-time switch 518
USE FOR DEBUGGING procedures 521
source listing 50

source name 53
source program 267
  batch entry 22
  compiling 37
  COPY statement considerations 30
  entering 21
  interactive entry 21
  library feature 30
  using SEU to enter 21
  WITH DEBUGGING MODE switch and compilation 518
SOURCE-COMPUTER paragraph
  DEBUGGING MODE as compile-time switch 518
  format 269
  syntax checker restriction 24
  treated as documentation 271
space fill
  example using INSPECT 436
  in an elementary MOVE statement 439
  punctuation character 6
space separator
  rules for using 13
SPACE/SPACES figurative constant 11
SPACE(S) 11
spaces
  BLANK WHEN ZERO clause causes insertion 328
spacing
  of pages and LINAGE clause 306
  program 28
special collating sequences, specifying 272
special display screen formats 22
special features
  debugging 517
  inter-program communication 507
  segmentation 503
  sort/merge 491
  table handling 469
special insertion editing 339
special level-number concepts 312
special registers 10
  DATE 375
  DAY 375
  DB-FORMAT-NAME 130, 373
  DEBUG-ITEM 523
  IBM extension 11
  TIME 376
special-character word, definition of 11
SPECIAL-NAMES paragraph 230
  ACCEPT statement 373
  alphabet-name clause 276
  CURRENCY SIGN clause 277
  DECIMAL-POINT IS COMMA clause 277
  example 274
  format 271
  function-name-1 clause 272
  function-name-2 clause 273
  PROGRAM COLLATING SEQUENCE clause 272
  syntax checker restriction 24
specific file processing 229
specification order
  data description clauses 317
  data-name or FILLER clause 317
  level-number 317
  REDEFINES 317
  subscripts 472
specifications
  for externally described files 216
  record format 214
split-end display 48
spool
  input 208
  output 208
spooling files
  inline data files 208
  output files 208
spooling function 22
square brackets.
  optional 14
  use of 38
stacker selection 535
standard alignment rules
  alphabetic items 313
  alphanumeric items 313
  JUSTIFIED clause modifies 328
  numeric edited items 313
  numeric items 313
standard COBOL format 24
standard data format 314
  numeric literal size 8
STANDARD phrase of LABEL RECORDS clause 305
STANDARD-1 phrase
  of alphabet-name clause 275
  SEQUENCE phrase 497
START statement
  access considerations 408

START statement *(continued)*
device considerations 408
format 407
generic 237
INVALID KEY phrases 409
organization considerations 408
relative key 289
sets current record pointer 372
starting line number
duplicate record keys, DUPLICATES phrase 289
formula 141
WRITE statement, for TRANSACTION file 140
STARTING phrase 141
statement 5, 347
ACCEPT 126
ACQUIRE 127
categories 349
CLOSE 128
COBOL, CALL 59
conditional 349, 368
copy 30
DROP 129
generic START 237
imperative 349
OPEN 129
procedure branching 20
PROCESS 37, 46
READ 131
REWRITE 138
USE 145
WRITE 140
statement number 53, 57
compiler-generated 50
statement-to-statment transfers 20
statements
consecutive 20
phrase 5
Procedure Division 5
rules for use 5
sentence 5
statements and clauses, summary of 567
static values of a table 475
statistics, compilation 49
status key values 548
STATUS KEY, file processing
OPEN statement 393
use 370
WRITE statement 414
STOP RUN statement
description and format 467
inter-program communication 262
STOP statement 467
ALL literal figurative constant restrictions 11
format 467
stopping points 88
storage
auxiliary 543
storage allocation, calling and called programs 509
storage format, USAGE clause specifies 322
storage layout of table, example 470
storage of records
illustrated 312
REDEFINES clause and 319
storage required 38
STRING statement 5, 441
ALL literal figurative constant restriction 11
examples 443
format 441
literals 7
nonnumeric literals 7
strings of characters 6
STRSEU, CL command 21
structures
data field 222
format (record) level 221
indicator 222, 223
subfield contents of DEBUG-ITEM special register 524
subfile
access 107, 132
special register DB-FORMAT-NAME 108
specified in DDS 106
use of 108
valid operations 107
SUBFILE phrase 131
subfile records
random access of 136
sequential access of 136
subject
of abbreviated combined relation-condition 363
of OCCURS clause, definition 476
of relation condition 356
Submit Job (SBMJOB) CL command 75
subordinate entries 16
subprograms 47

subscripting 472
and indexing 18
and PROCESS statement 473
example 473
invalid for File-Control entry data-names 284
of data-name 528
restriction for qualifiers 18
substitution field of INSPECT REPLACING 433
SUBTRACT statement
common phrases 422
CORRESPONDING phrase 428
formats 428
subtraction operator 353
successive comment lines 29
successive entries 28
successive sentences 28
summary of COBOL statements and clauses 567
suppress diagnostic messages feature 1
suppression of messages 542
suppression of sequence checking 25
suspension of running
ACCEPT statement 373
STOP statement provides 467
switch-status condition
format 359
switch, IBM-defined 19
switches, debug run-time 74
symbol order in PICTURE clause 335
symbols used in PICTURE clause 332
SYNCHRONIZED clause 327
format 327
syntax checking 37
syntax checking of card device instructions 533
syntax error, problem determination 87
syntax of program
debugging lines 524
syntax-checking feature 1
syntax, invalid use of 541
system console
ACCEPT statement 373
DISPLAY statement 386
system information transfer, ACCEPT
DATE 375
DAY 375
TIME 376
system input device, ACCEPT statement 374
system override considerations 209
system reply list 75
system-dependent considerations
Data Division considerations
BLOCK CONTAINS clause 303
system-dependent considerations *(continued)*
Data Division considerations *(continued)*
COPY DDS statement 219
index literals 474
item size 304
LINAGE clause 307
OCCURS clause 476
RECORD CONTAINS clause 304
SORT/MERGE statement 495
subscript literals 472
Environment Division considerations
ASSIGN clause 284
RECORD KEY clause 289
RESERVE clause 286
SAME AREA or SAME RECORD AREA clause 292
SAME SORT-MERGE AREA clause 493
general considerations
indexed file 288
library-name 17
program-name 268
relative file 288
source program library 30
source statements 22
text-name 17
user-defined words 9
Procedure Division considerations
arithmetic statements 420
CALL statement 510
GO TO DEPENDING ON statement 454
INSPECT statement 429
STOP statement 467
UNSTRING statement 445
system-error routine 366
system-independent binary items 324
system-name 9
SYSTEM-SHUTDOWN as function-name 274
System/36
accessing data files on remote 278
field definitions on remote 278
System/38 1
accessing data files on remote 278
differences from AS/400 571
environment 21
field definitions on remote 278
System/38 COBOL, description of 1
SYSTEM/38 file name
in ASSIGN clause 286
in COPY statement 31

SYSTEM/38 library name
and Library-name 31
in COPY statement 31
System/38-Compatible COBOL
compiler, calling 37
features of 1
Language level supported by 1
programming considerations 205

T

table
definition 469
length 477
table element
definition 469
length 477
table handling
Data Division 476
OCCURS clause 476
Procedure Division 480
reinitializing index-names 488
relation conditions 481
SEARCH statement 481
SET statement 488
table definition 469
table initialization 475
table references 471
UP/DOWN BY phrase 490
USAGE IS INDEX clause 480
table handling concepts
table definition 486
table initialization 475
table references
indexing 473
subscripting 472
Table Handling module, 1974 Standard 2
table layout, example 470
table of valid and invalid moves 439
table references
and SEARCH ALL results 484
indexing 473
subscripting 472
table values, defining 475
TALLYING phrase
INSPECT statement 433
UNSTRING statement 446
tape rewinding/unloading 529
techniques, displaying variables 67
TERMINAL phrase 131, 135, 137, 139, 140, 144
with READ (nonsubfile), description 139
with READ statement, formats 131
with READ SUBFILE, description 137
with REWRITE statement, format 138
with WRITE (nonsubfile), description 140
with WRITE statement (nonsubfile), format 140
with WRITE SUBFILE, format 144
termination of processing
EXIT PROGRAM statement 513
STOP RUN statement 467
termination, program 60
test library 61
testing a program 62
testing function, OS/400 60
text-name
COPY statement operand 30
formation rules 9
qualification format 16
THEN phrase
format 367
used as separator 367
TIME, ACCEPT statement 373
TIMES phrase of PERFORM statement 458
TO phrase, SET statement 488
top page margin in LINAGE clause 307
traces 63, 71
considerations 73
example 72
tracing a loop 255
TRAILING phrase of SIGN clause 326
TRANSACTION files
Boolean data facilities 126
considerations for 530
data description specifications for 90
Data Division considerations 125
Environment Division considerations 122
example programs 145
externally described 90
language extensions for 89
Procedure Division considerations 126
processing externally described 92
program described 122
TRANSACTION organization 123
transfer of control
ALTER statement change 453
and sort/merge OUTPUT PROCEDURE 499
explicit and implicit 20
sort INPUT PROCEDURE 498

transfer of data
in a STRING statement 442
into DEBUG-ITEM special register 523
transfers of control 20, 530
transfers, statement-to-statment 20
truncation
in numeric items 313
JUSTIFIED clause 328
truncation of data
ACCEPT statement 373
in an elementary MOVE statement 439
in floating insertion editing 341
incompatible record lengths 305
ROUNDED phrase 423
VALUE clause restrictions 329
truth value, 360
twos complement form 325

U

unary operator
list 353
use 353
unattended mode, running the program 542
unblocked files, BLOCK CONTAINS clause 303
unblocking code, generation of 552
unblocking input records/blocking output records
conditions 210
file status values 291
OPEN statement 393
updating the I-O-FEEDBACK area 552
unblocking, automatic 370
unconditional GO TO statement 454
underscores, translated to hyphens 220
undesirable results, when using index as subscript 66
unequal level-numbers 528
unexpected results, problem determination 87
unsigned numeric integers 18
unsigned numeric literal considered positive 8
unsigned operand
considered positive or zero 314
UNSTRING statement
data receiving fields 446
examples 450
format 445
processing rules 447
sending field 445
UNTIL phrase of PERFORM statement 458
UP/DOWN phrase, SET statement 488
UPDPROD(*NO) parameter, ENTDBG CL command 61
UPON phrase of DISPLAY statement 386
UPSI (User Program Status Indicator) 575
UPSI switches
and SET statement 440
and switch-status condition 359
SPECIAL-NAMES paragraph 274
UPSI-0 through UPSI-7 as function-names 273
USAGE clause 94, 322
and numeric items 323
computational options 323
DISPLAY phrase 322
format 322
operational sign representation 314
REDEFINES clause and data values 321
zoned decimal items 323
USAGE IS INDEX clause 480
format 480
USE statement 145
EXCEPTION/ERROR 365
EXCEPTION/ERROR for TRANSACTION file 145
FOR DEBUGGING 521
User Program Status Indicator (UPSI) 575
user-created command 59
user-defined word
characters in 9
formation of 9
formation rules 9
hyphens in 9
representation in manuals 9
rules for using 9
user-name
as function-name 9
in VALUE OF clause 306
user-specified name 15
uses of a character string 7
USING phrase
SORT/MERGE statement 497
USING phrase, inter-program communication format 511
using REPLACING with format 2 COPY 226
using separators 13

V

valid and invalid elementary move table 439

valid characters in COBOL
ascending EBCDIC sequence 6
valid characters in CURRENCY SIGN clause 277
valid COBOL characters, IBM extension 6
valid processing sequence, PERFORM statement 457
valid RECORD KEYS 237
validity checking 90
validity checking, automatic 370
VALUE clause 95
example of condition-name entries 331
format 328
VALUE OF clause 306
value, of numeric literal 8
variable length table 477
format 477
variable, conditional 317, 330, 355
variables, techniques for displaying 67
varying operands in PERFORM statement 459
VARYING phrase
PERFORM statement 459
SEARCH statement 481
verb usage by count listing 52
verb usage listing 49
verbs
as keyword 10
lists 351
violation 280, 414
violations flagged, FIPS 55

W

WHEN phrase of SEARCH ALL statement 483
WITH FOOTING phrase of LINAGE clause 307
WITH NO REWIND phrase of CLOSE 379
word 15
in COBOL 8
key 14
optional 10
reserved 14
user-defined 9
words, reserved 563
work station 145
work station support 89
See also TRANSACTION files
Working-Storage Section
general description 299
general format 296
level-77 and level-01 names unique 312
VALUE clause considerations 329
WRITE ADVANCING statement 416
LINAGE clause and 308
WRITE statement 412
access considerations 414
ADVANCING phrase 415
device considerations 414
END-OF-PAGE phrase 415
for TRANSACTION file 140
format 412
FROM identifier phrase 371
INVALID KEY phrase 415
mnemonic-names and 273
modifies LINAGE-COUNTER 308
organization considerations 414
ROLLING phrase 141
STARTING phrase 141

Z

ZERO
as Boolean literal 12
as figurative constant 11
zero (0)
as unique value 314
insertion symbol 333
zero filling
INSPECT statement 436
zero suppression and replacement editing 342
ZERO, ZEROES, ZEROS figurative constant 11
used as Boolean literal 126
ZERO(S)(ES) 11
zoned decimal item 323
RECORD KEY data item 290
zoned decimal items, conversion to 529

IBM®

Program Number: 5763-CB1

Printed in U.S.A.

SC09-1814-00